INTERNATIONAL
HELLENIC
UNIVERSITY

Landmark and monument recognition with Deep Learning

Theodoros Stougiannis

SID: 3308190025

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

JANUARY 2021

THESSALONIKI – GREECE

INTERNATIONAL
HELLENIC
UNIVERSITY

Landmark and monument recognition with Deep Learning

Theodoros Stougiannis

SID: 3308190025

Supervisor:	Prof. Konstantinos Diamantaras
Supervising Committee Members:	Assoc. Prof. Name Surname
	Assist. Prof. Name Surname

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of Data Science

Master of Science (MSc) in Data Science

JANUARY 2021

THESSALONIKI – GREECE

Contents

Acknowledgement

I would like to thank my supervisor, Professor Konstantinos Diamantaras for encouraging my interest and for his valuable guidance through the steps followed in order to achieve the aims of this dissertation.

Abstract

With the recent advancements in deep learning, we have the potential to train complicated machine learning models that involve identifying various objects in an image irrespective of their attributes or the different perspectives. In this thesis, we assess the feasibility of utilizing deep learning models to recognize objects in real-time on mobile devices with respect to performance and inference time. More specifically, our aim is to train a model based on the monuments of the UNESCO Monuments Route of Thessaloniki with Tensorflow Object Detection API and consequently deploy it into Android with Tensorflow Lite. SSD MobileNet V1 and SSD MobileNet V2 were trained on a collection of personal and images from the internet using Google Colab and Transfer Learning. Additionally, we introduced the technique of data augmentation in order to enhance the performance of our model.

The results are particularly encouraging with SSD MobileNet V1 achieving an overall accuracy of 93.46% and the SSD MobileNet V2 with an overall accuracy of 95.66%. Applying data augmentation on the latter of the two models we were able to accomplish a slight increase in the accuracy reaching 96.25%. Finally, the findings in the present work confirm the fact that deep learning can be used to train a model and apply it to a mobile device, offering quite satisfactory outcomes even if the model is trained on a small and with low-quality data.

Keywords: Deep Learning; Monument Recognition; Single Shot MultiBox Detector (SSD); Transfer Learning; Tensorflow Lite; Android;

Theodoros Stougiannis

04 January 2021

1 Introduction

Computer vision constitutes a multidisciplinary scientific field that deals with the way of how computers comprehend or perceive the visual world. In other words, how computers can identify and classify an object, based on a compilation of images or videos used for input [1]. As humans, we have the ability to understand the three-dimensional structure of our surrounding environment. In the same way, we try to imitate this procedure in computers, using mathematical techniques to create a partial 3D model of an object [2]. Nowadays, computer vision is utilized in a great variety of different sectors like Optical Character Recognition (OCR), Machine Formal Review, Retail, Motor Vehicles Safety, Surveillance, 3D modeling, Face Recognition, and many more [3]. Of course, nothing of these would have been achieved without the recent advancements in the field of machine learning and more specifically in deep learning. With the advent of deep learning and the recent developments in the last few years, we now have the potential to build complex Machine Learning models for image or object detection without considering the characteristics of an object [4].

Nonetheless, the most substantial part that led to these outcomes is well known with the name Convolutional Neural Network (CNN). Plainly, a Convolutional Neural Network which belongs to the family of Neural Networks is a Deep Learning algorithm that takes an image as an input, it allocates significance(weights) to multiple diverse aspects of objects in the image and tries to identify and discern one from the other [5]. Taking into account the last trait, we may infer that Convolutional Neural Networks could also offer improved results when it comes to Landmark/Monument recognition [6].

Without any doubt, Landmark/Monument recognition constitutes a significant feature, not only for tourists who visit various places around the world but also for the local people in order to better learn and get informed about their cultural heritage. In this way, someone has the opportunity to install an application to his personal smartphone, open the Landmark/Monument recognition application, and obtain information about the Landmark/Monument which is right in front of him, even if there are no guides there [7].

At the same time, this method of learning offers an interactive character making the process possibly more interesting and stimulating.

Definitely, the rapid growth and the radical developments of the technology industry especially in mobile devices played a fundamental role in on-device image recognition and object detection. As mobile phones consist an inextricable part of our lives and our demands have led to devices with comparatively much higher performance, the past few years we have the chance to apply such Machine Learning algorithms to image recognition and object detection on these devices [4][7].

When implementing these types of technologies, however, we must consider the computational and generally the hardware power of smartphones as may be inadequate. For this reason, as computational saving is vital, it is of paramount significance for the operation to be optimized [4]. This is where Tensorflow comes into bridging the gap between the Machine Learning algorithms and the limited hardware power of mobile devices. Tensorflow is an open-source library used for Machine Learning offering several neural network models [8]. Particularly with the Tensorflow Mobile, researchers and developers have the ability to pick the desired model, deploy it even on a low-end device and apply an optimization procedure to further enhance the smooth operation of the application [9].

The goal of this thesis is to implement and deploy a Deep Convolutional model for image recognition to recognize the Landmarks/Monuments.

1.2 Scope

The thesis entails the creation and the development of a Deep Learning application on Android that can detect Landmarks and Monuments in real-time and from different views. The tested Landmarks/Monuments consisted of the UNESCO World Heritage Monuments of Thessaloniki. At this point, it is considerable to mention that the recognition should allow the possibility of the various lighting conditions, different points of view as well as the potential partial occlusions. The fact that the geometrical shape of the desired objects is often similar to many other objects and a minor change of the angle may provide totally different information, renders this task challenging. These challenges can be identified at several stages of this dissertation.

1. When we refer to image recognition or image processing, the computational cost is something that should be seriously considered. Perhaps, the most vital task is to deal with both the time needed to train the model and inference time required to produce the predicted outcome, given the limited capabilities of the mobile devices.
2. A time-consuming but very crucial stage of the process is the gathering and the preprocessing of the training data. Selecting the appropriate images for the model along with the implementation of the most suitable preprocessing techniques might alter dramatically the performance or the accuracy of the model.
3. The selection of the proper model is also a challenging but again significant decision. Often, we must determine between speed and performance. Some models are very fast but do not provide very accurate results and others offer exceptional results with low speed though. Additionally, there are models that generally perform better in specific tasks or certain fields.
4. After training a model, in order to feed it into the Android platform, you are required to convert the model to the appropriate form. Apart from this, to ensure the smooth operation of the application, optimizing the model is imperative and at least recommendable.

1.3 Thesis outline

The structure of the presented thesis is divided into several chapters as follows:

- Chapter 1: Introduces the subject of the research and the main question that is being investigated along with the basic challenges that are attached with it.
- Chapter 2: Contains the most significant and most recent related work conducted in the field. In each separate case, the procedure followed and the essential parts of the work are summarized.
- Chapter 3: The main purpose of this chapter is to provide the theoretical knowledge that is crucial for the better comprehension of not only the various concepts behind the main research subject but also the reasoning for the implementation of the desideratum.
- Chapter 4: Entails the contribution and the conducted experiments. Alternatively stated, includes the principal process followed in order to fulfill the given goal. The chapter consists of 12 stages that are necessary to implement the central aim.

- Chapter 5: Demonstrates the results of the experiment.
- Chapter 6: Presents some important to the thesis issues and thoughts.
- Chapter 7: Contains suggestions of how to further develop the conducted work.
- Chapter 8: Entails the epilogue of the provided thesis.

2 Related Work

Recent developments in Deep Learning and especially in Convolutional Neural Networks which are remarkably well with computer vision, have created many opportunities and finds application in multiple fields. Concurrently, advancements in the mobile industry have led several researchers and developers to explore, combine these two sectors and exploit the chance for utilizing and optimizing techniques for object detection on mobile devices. One usage of this technological innovation, which is particularly valuable for people who travel around the world and meet new places, is that of the monument or landmark recognition. A practical and effective way, of learning and studying foreign or the local cultural tourist attractions, just by installing a single application in the personal smartphone.

In 2017, Siddhant Gada, Viraj Mehta, Karan Kanchan, Chahat Jain, and the Prof. Purva Raut published a paper with the title “Monument Recognition using Deep Neural Networks” [5]. In their project, they try to classify well-known Indian Monuments located in the Golden Quadrilateral area, using a famous deep learning architectural model, the Inception v3. This architectural structure encompasses multiple diverse layers having the peculiarity of constantly processing the dataset offering a more improved model. A crucial feature of the successful train of this neural network is the activation function, in this case, Rectified Linear Unit (ReLU), which offers smooth performance and improved training time. In addition, they introduce the notion of pooling and more specifically the max pooling, a prominent technique in machine learning that summarizes the presence of features in an input image [8], making a model more robust. Collecting approximately 400 pictures for every monument and using a retrained model, after 4000 iterations they were able to achieve a cross-entropy close to 0.067, a training accuracy of 99.4%, and a corresponding testing accuracy estimated from 96-99%.

Two years later, V.Palma, at the “A Monument Recognition Mobile App” [6], attempts to develop a mobile app that is able to identify and recognize monuments. The basic notion behind the application relies on deep learning methods and in particular the convolutional neural networks. Utilizing convolutional neural network techniques, the app has the capability of recognizing architectural objects. To elaborate a bit more, the aforementioned mobile application is consisted of two main sectors, at the first one you upload the desired document and possibly related data, and at the latter of the two sectors, the application takes advantage of machine learning software to provide information on monuments stored in an online database. He develops two separate applications, one for the Apple iOS and the other for Android OS. Providing 50-100 pictures for the 46 different monuments and exploiting data augmentation techniques to enhance the dataset (500 pictures per monument), the model can correctly recognize the desired monument with an accuracy of 95% or more.

Another significant contribution regarding the monument/landmark recognition was that of Chakkrit Termritthikun, Surachet Kanprachar, and Raisarn Muneesawang, at the "NU-LiteNet: Mobile Landmark Recognition using Convolutional Neural Networks" at 2018 [7]. They understand that although we can achieve accuracy comparable to that of a human, realistically, there is the hindrance of the latency. In their paper, they construct a new convolutional neural network model, based on the development idea of SqueezeNet [9], with the name NU-LiteNet. To achieve their task, they build two different forms of NU-LiteNet, the NU-LiteNet-A and the NU-Lite-B, both with two additional convolutional filters of 5x5 and 7x7 to enhance the accuracy score, but with the dissimilar amount of depth. The experiment was conducted on two datasets, the first one feed with 4060 images for 50 landmarks of Singapore and the second one feed with 6412 images for 12 landmarks of Paris, in France. After splitting the dataset for training and test at 90% and 10% respectively and resizing the images to 256x256, they compare the results with these of the AlexNet, GoogLeNet, and SqueezeNet. Using the top-1 as well as the top-5 accuracy to measure the performance, they achieve higher accuracy even that from GoogLeNet with 7.4-10.46% higher for the Singapore dataset and 6.7-961% for the Paris dataset, considering at the same time the reduced size of the model by 2.6 times in comparison with SqueezeNet. Furthermore, to fulfill their goal they develop an application on Android OS, feeding it with the aforementioned models. The results reveal the NU-LiteNet-A as the most skillful model with just 1.07 MB and processing time of 637 ms per image.

The same year, Ajay Kumar Mallicka, Aniket Ninawe, Vikash Yadav, Hifzan Ahmad, Dinesh Kumar, Sah and Cornel Barna attempt to solve an image classification problem, classifying whether an architectural structure, a monument, belongs to the Cathedral or the Indian Mughal, “Cathedral and Indian Mughal Monument Recognition using Tensorflow” [13]. A deep learning technique is followed to train a convolutional neural network to be found on Tensorflow, an open-source library used for machine learning. To conduct the research, they make use of 1000 monuments of both Cathedral and Indian Mughal, with different types of monuments to effectively allow the model to learn possible variations between monuments of the same class. Exploiting the convolutional neural networks to extract general features and obtain the train weights, they manage to achieve 80% accuracy, performing better than Gupta et al. [14] by more than 7%.

An also appealing approach was that of Aradhya Saini, Tanu Gupta, Rajat Kumar, Akshay Kumar Gupta, Monika Panwar, Ankush Mittal with the “Image based Indian Monument Recognition using Convoluted Neural Networks” published back in 2017 [10]. They identify that monument recognition is a challenging task, where you have to take into consideration the huge variations and the diverse orientation of the monuments. The basic approach behind their attempt is relying on the features that you can obtain from the monuments. To recognize the monuments, they extract features using hand-crafted and convolutional neural network methods. The key concept behind the monument classification based on the hand-crafted features entails the Histogram of Oriented Gradients (HOG), the Local Binary Patterns (LBP), and the (GIST) features techniques. For the experiment, they utilized 50 images for every one of the 100 different monuments located in India. Various strategies were conducted in combination with famous classification algorithms, just like Support Vector Machine (SVM), k-Nearest Neighbor (KNN), and Random Forest. Findings reveal superior performance for convolutional neural models over the hand-crafted cases, achieving significantly better accuracy measured at 92.7%. Moreover, they introduce the concept of Graph-Based Visual Saliency (GBVS), a tool able to pinpoints the salience of an image providing better results [11], [12].

3 Theoretical Background

In order to better comprehend the idea and the aims of this thesis, it is essential to look into some key variables as well as the relevant data around our framework.

3.1 History of Computer Vision

As aforementioned, computer vision is the attempt of humans to translate the visual world into a model comprehensible by computers.

Computer Vision was first made an appearance in the late 60s, as a stepping stone for an artificial intelligent project, where scientists trying to imitate the operation of the human visual system [15]. During this period, by some early optimistic pioneers on this section, mainly at universities, it was assumed that processing data from images to extract a 3D geometrical model, well known as the “visual input” problem, would not be a difficult task [2]. Soon enough researchers realized that obtaining a 3D structure from digital images was a complex and challenging assignment.

In the 1980s, studies focusing on meticulous mathematical techniques and quantitative aspects of an image or a scene. This perception entailed techniques such as image blending, scale-space processing, displacing or augmenting, shading, contour models, and many more [16], [17], [18]. Another noteworthy approach that could probably benefit the research was using the Markov Random Field (MRF) models [19].

During the 1990s and when the above-mentioned methods were continued being investigated projective reconstructions and factorization techniques will contribute to the evolution of computer vision [2]. Camera calibration optimization techniques and already existed knowledge on the field of photogrammetry will lead to formulas for constructing a scene based on multiple images [20]. In addition, in the same decade, image segmentation and statistical learning techniques along with the increased interaction with computer graphics will play a fundamental role.

The 2000s was the decade, were trends like texture synthesis, feature-based techniques and the development of more capable and effective algorithms turned the tide. Notwithstanding, perhaps the most essential trend was the advent of machine learning, offering to the community the opportunity to classify objects without human supervision, but just with the vast amount of labeled data accessible via the internet [2].

Nowadays, taking into consideration the enormous amount of visual data, the even more efficient algorithms, and adequate computational power, the field of computer vision has grown dramatically. In such a small period of time, today's models are able to identify and pinpoint an object with tremendous accuracy, even higher than this of a human [21]. The capabilities that computer vision has created in the past few years are limitless and many fields have radically changed. Concrete examples include the automobile sector with the emergence of self-driving cars [22], the Google Translate application able to translate a text just by pointing a phone's camera [23], facial recognition [24], the healthcare sector with medical diagnostic methods based on images [25] and many more (Figure 1).

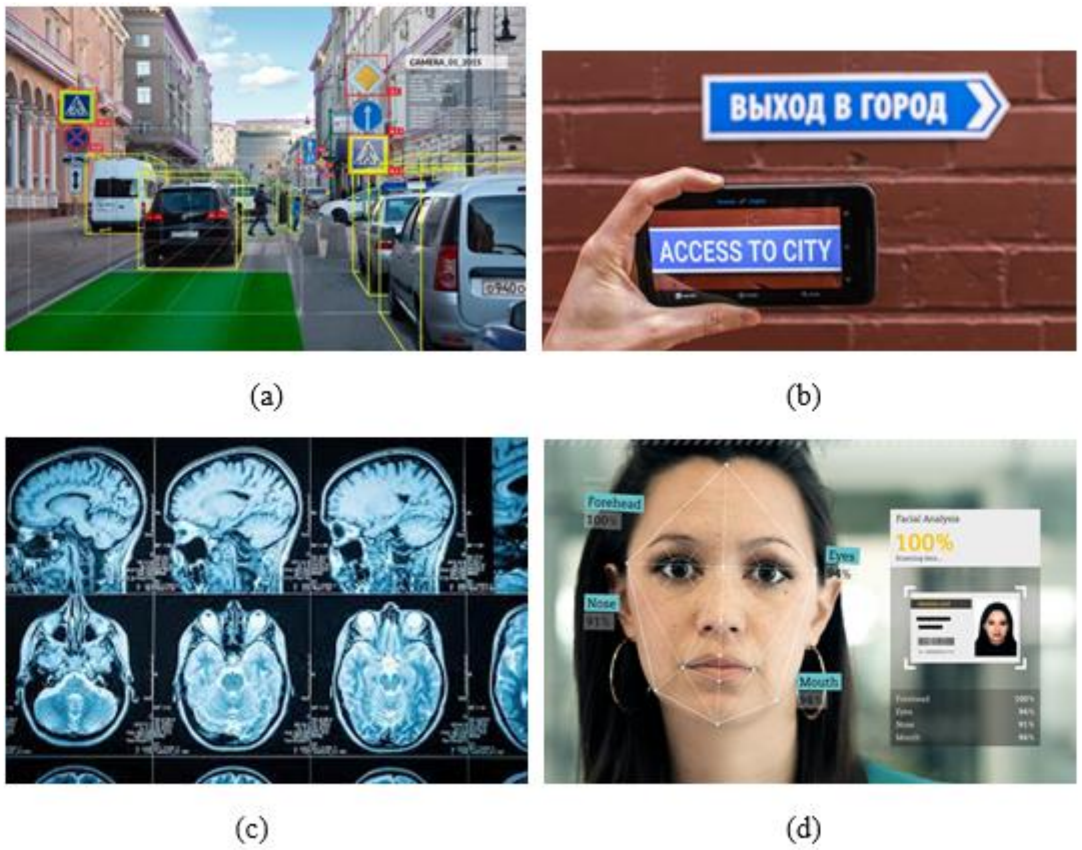


Figure 1: Industrial utilizations of computer vision. (a) Self-driving car interface. (b) Google translate app translates though a phone's camera a sign in the streets. (c) Machine learning supports diagnosis for neurological disorder. (d) Machine learning can identify a face as well as basic parts of the face.

3.2 Machine Learning and Deep Learning

In this day and age, there is a plethora, a vast amount of data produced from personal computers, smartphones, sensors, and other sources and which has altered and reshaped the way we deal with a variety of modern problems. For some of these challenges, we use algorithms to solve them, nonetheless, there are other challenges where we do not have specific algorithms and different kinds of solutions are required [26]. This is where Machine Learning comes to bridging the gap. Machine Learning constitutes a part of Artificial Intelligence and is the study of computer algorithms that enhances, adapt to the outer environment, and learns through experience automatically. The "learn" component is the core concept of this approach. If a system has the skill to learn and adapt to a changing environment, then the designer does not have to search and provide a solution for every possible outcome [27]. Commonly, machine learning algorithms are categorized into supervised, unsupervised, and semi-supervised algorithms. The first category refers to the case where a priori knowledge is provided, often in the form of class labels. In brief, in this case, models learn from labeled data and then are used to make a prediction based on what they have learned [28]. In contrast, in unsupervised learning, no previous knowledge is granted. Typically, unsupervised algorithms are applied in order to pinpoint patterns, trends, or usual observations [29]. As the name suggests, semi-supervised learning is something between supervised and unsupervised learning. Usually, a semi-supervised method is used when most of data is unlabeled and only a small portion of it is labeled. Apart from the above categories, in some cases, we can also use a method well known as reinforcement learning. It has many similarities with semi-supervised learning, however, it is mostly being utilized in robotics, gaming and navigation where a number of actions are applied, and only at the end of the policy (the whole number of actions) an agent (decision maker), characterize this policy as good or not good. Implementing a strategy of trial and error, an algorithm tries to identify the best possible policy, taking into account every time the outcome of the prior trials [30].

Deep learning or structured learning is a subfield of machine learning using algorithms that incorporate the concept of artificial neural networks. The idea stems from the biological neural networks [31] that we encounter on animal brains. In common, a biological neuron consists of a number of distinct organs containing dendrites, nucleus, synapse, and axon each one responsible for a specific task. This exact way of operating we try to imitate with artificial neural networks, using an input which reproduces the function of dendrites, weights to represent the synapse stage, an activation function as the nucleus, and finally an output to mimic the operation of the axon [31], [32]. In (Figure 2) we can observe the structure of the biological neural network along with the matching artificial parts.

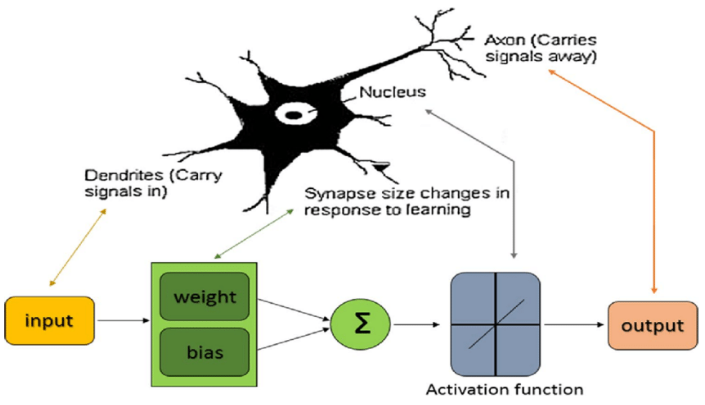

Axon (Carries signals away)
Nucleus
Dendrites (Carry signals in)
Synapse size changes in response to learning
input
weight
bias
Σ
output
Activation function

Figure 2: Structure of biological neural network with the corresponding artificial components.

In a nutshell, an artificial neural network consists of multiple nodes, known as artificial neurons, that are connected and interact with each other. In every connection of the artificial neurons, just like with synapses in a biological brain, a signal is transmitted through the neurons. A simple operation, called activation, is conducted in each node, passing the results as an input in the next neurons. Furthermore, each connection is associated with weights, which are being updated in every iteration offering the artificial neural network the ability to learn. To better understand the described procedure, an illustrative example is presented in (Figure 3) [32], [33].

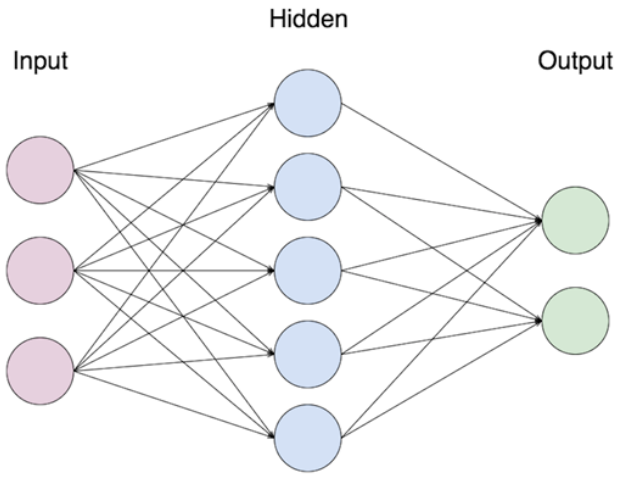


Figure 3: Structure of artificial neural network.

Although, in machine learning, a plain mathematical model is able to learn from the data and provide a prediction, in the case of deep learning an algorithm contains more than one hidden layer between the input and the output, often a significant number of them [30]. Regularly, in deep learning, it is essential to train a model with a large amount of data which in turn will give back the analyzed features without the need for manual guidance. Some of the most popular deep learning architectures, include Deep Neural Networks (DNN), Recurrent Neural Networks (RNN), Convolutional deep Neural Networks (CNN), and Deep Belief Networks (DBN) and are applicable especially in fields such as computer vision, speech recognition, natural language processing, social network filtering, medical inspection and several more [34].

Convolutional neural networks are of particular interest for this dissertation, as they perform remarkably well with image and video recognition, as well as with image analysis and classification. Therefore, convolutional neural networks are being analyzed in the following subsection.

3.3 Convolutional Neural Networks

A convolutional neural network (CNN/ConvNet) is a type of feed-forward artificial neural network, able to take a picture as an input, allocate significance to the different viewpoints or objects in the image and then distinguish the various included entities [30]. Inspired by the organization of Visual Cortex, the structure of a convolutional neural network is alike to that of the human brain. Incorporating this type of architecture, a CNN model is able to obtain both the spatial and timing dependencies presented in an image, offering the ability to comprehend even the most demanding images [35].

Images can be illustrated as a 2- dimensional grid structure, where every point matches a pixel. In an RGB image, every pixel possesses three values of intensity, each one for the three primary colors, red, green, and blue. Considering that a picture has spatial dimensions of 100 x 100, subsequently, given the three different colors, the number of variables will be equal to 100 x 100 x 3 = 30,000. Immediately, we can assume that for a high definition picture (1080 x 1024) the computations

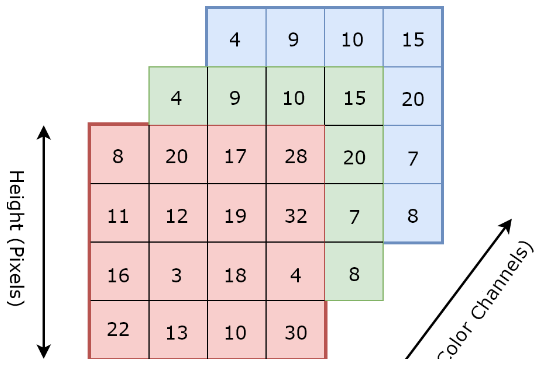


Figure 4: 4 x 4 x 3 RGB Image

are considerably increased. The main purpose of the ConvNet is to downscale the images to an effective form, considering at the same time the significant parts and features to maintain a pleasant prediction [36]. In the following (Figure 4) we can notice an RGB image, with dimensions 4 x 4 x 3.

A ConvNet may be comprised of various blocks including the convolutional layer, the nonlinearity layer, the pooling layer and the fully connected layer which are introduced right away (Figure 5). Each layer is responsible for a particular task and is important to understand how exactly it functions.

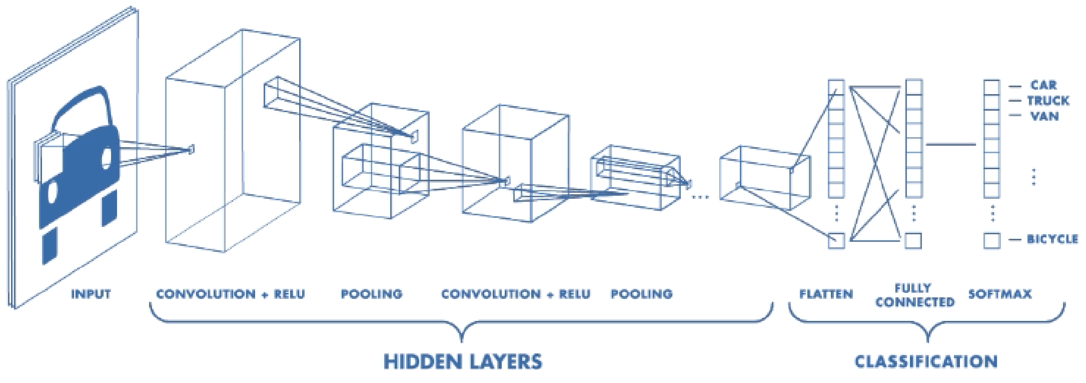


Figure 5: An example of a convolutional neural network's structure, consisted of various layers like convolutional layers, non-linear layers, pooling layers and fully connected layers. The architecture of the layers and the way that they are utilized may vary.

3.3.1 Convolutional Layer

The first layer in CNN is called Convolutional Layer. Very simply, a Convolutional Layer is composed of a set of filters and its role is to carry out a convolutional operation among the filters and the input in order to construct feature maps. In other words, this is the stage where we try to extract features from an input, using grids of discrete numbers (filters), often with squared shapes. Utilizing the elements of the input data and parameters provided from the filter, it will supply as with the summation value, as it is also illustrated is the (Figure 6) [36], [37].

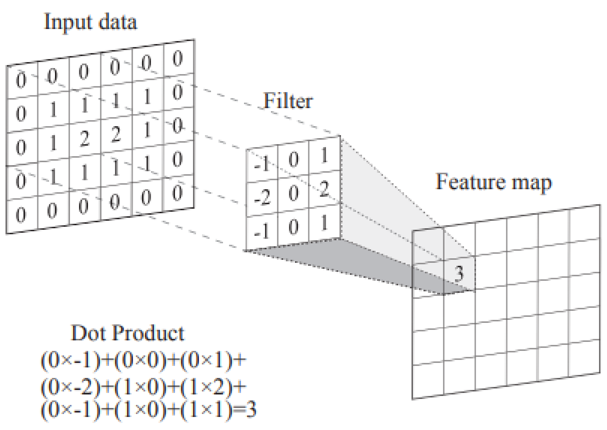


Figure 6: Convolutional Operation. Each element in the feature map is produced by multiplying the convolutional filter with the corresponding grid from the input data.

Applying the same procedure for the rest of the grids in the input data, eventually, we will get the entire feature map (convolved features). Different filters can be provided to give a totally different aspect of the image.

An essential factor of the convolution operation is the “stride”. With the term stride, we refer to the number of dots or pixels that we desire to slide the convolution filter over the input matrix. For instance, when the stride value is equal to 1, then we shift the filter from pixel to pixel. Following the same rationale, it is safe to speculate that with a larger stride we tend to have a smaller feature map [38].

In some cases, there is a possibility, for the filter to not absolutely match the input image. Therefore, it is practical to pad the image with zeros around the border, covering the parts that do not fit. This technique is popular with the name “zero-padding”. However, another useful method is the “valid padding”, where we drop some pieces of the image to aid the convolutional filter to fit the image.

After every convolutional layer, in order to regulate the output of every layer, we use an additional layer that incorporates a non-linear function, called “Activation Layer” [39]. This is the layer responsible to determine whether a part of the input will participate in the class label or not. Alternatively stated, if a part of the input is important, then the activation layer will assign the corresponding significance. Primarily, the most usual types of non-linear methods followed contain Rectified Linear Unit (RELU), Sigmoid, and Tanh. The most frequently used non-linear operation is the RELU. The significance of the RELU, lies in its purpose, to wipe out all the negative values in the feature map and replace them with zero, as it is provided in (Figure 7).

Considering that the majority of real-world data produced are constituted from non-linear data, this is a vital task. Furthermore, this method allows us to train faster than the other aforementioned techniques, since the gradient, in this case, it does not take values close to zero, saving crucial time in the optimization stage [36], [39]. The output has the form: *f(x) = max(0, x).*

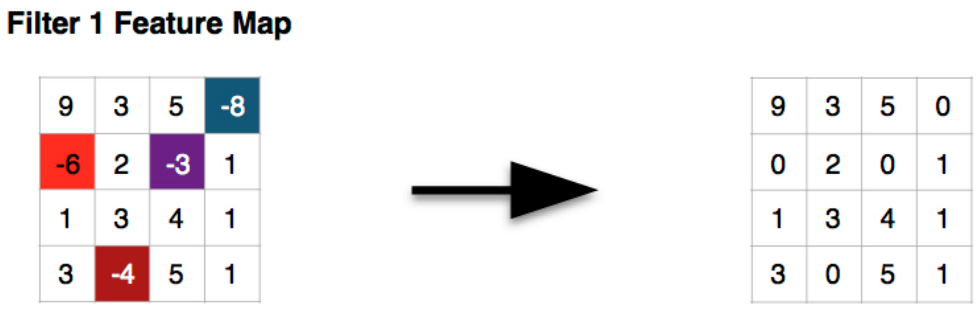


Figure 7: Feature map before and after the execution of the RELU function.

3.3.2 Pooling Layer

The aim of the pooling layer is to decrease the size of the feature map, enabling smaller and more manageable calculations. In other words, the pooling layer simplifies the information contained in the feature map and rebuilds it, maintaining the most useful parts. This operation is called sub-sampling or down-sampling and may have different types, such as max, average, or sum. Max pooling divides the feature map into a number of windows and receives the maximum element for each one of them, discarding the remaining values. The same technique is adipted in the case of average or sum pooling. Notwithstanding, max pooling is the most famous one, and in practice seems to work more effectively [36], [40]. In (Figure 8) we can observe an example of max pooling.

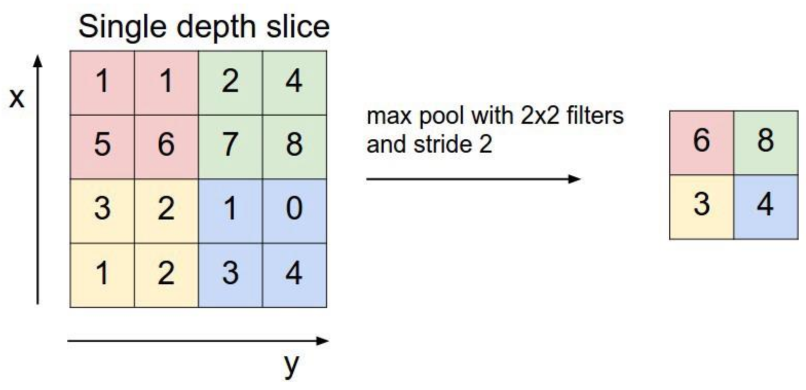


Figure 8: Example of max pooling operation.

3.3.3 Fully Connected Layer

Typically, the last part of CNN consists of fully connected layers. The purpose of these layers is to carry out classification, based on the features discovered and obtained from the previous layers. The term “fully connected” stems from the fact that each neuron in the previous layer is linked to each neuron on the following one [36].

It is common, for convolutional neural networks to be constructed by several convolutional and pooling layers. Often, more layers are added, to increase the density sufficiently so as to deal with more complex and non-linear problems, using at the same time a softmax layer as a top layer [41]. As mentioned above, (Figure 5) summarizes a convolutional neural network, following the described structure.

There are several types of famous convolutional neural networks that present their own special architecture and this specific structure can offer exceptional results. Some of the most popular are LeNet, AlexNet, VGGNet, GoogLeNet, ResNet, and DenseNet [42]. Each model uses its own build, with a different number of layers and a unique design.

3.4 Classification

Before proceeding with the Object Detection, it is essential to comprehend what exactly the term “Classification” means, as well as when it is being used. Classification is the procedure of specifying and categorizing a given input x into the class k that it belongs to. Regularly, this is achieved by using a function with the form, $f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$. So, the model allocates the input usually with the form of vector x to a class, providing an output with the predicted class y or with a vector Y which contains the probability distribution for the whole set of classes k [43]. For instance, in a classification process for object recognition, we have an image as an input and the expected results are a single category or the probabilities for the identified objects.

3.5 Object Detection

Object detection is a common challenge for Deep Learning and Computer vision. Its aim is to locate and classify objects identified in an image and create bounding boxes with the according labels, for each one of them [44] (Figure 9). As we can also observe from the example picture, object detection can be even applied to multiples objects and provide information for all of them.

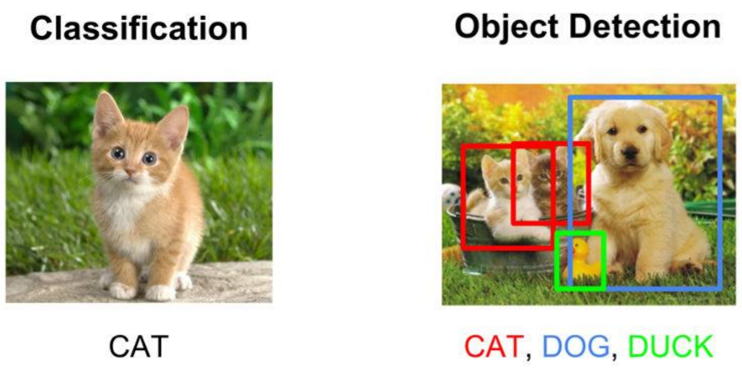


Figure 9: Demonstrating the difference between Classification and Object Detection.

Regularly, the process of object detection consists of some specific steps. Initially, the chosen algorithm pinpoints and creates important regions known as region proposals, composed of several bounding boxes that cover the interesting parts (Region Proposal Network) [44]. Sequentially, optical features are extracted for each one of the created bounding boxes and assessed in order to decide the possible objects in these regions. Eventually, the overlapping boxes are added all together to give a single bounding box. These algorithms can be characterized as Two-Step Object Detection algorithms and include models such as RCNN [46], SPPNet [47], Fast RCNN [48], Faster RCNN [49], and Feature Pyramid Networks [50].

Nevertheless, extensive demands for real-time object detection have led also to the construction of one-stage architectures. Such architectures contain models like You Only Look Once (YOLO) [51], Single Shot MultiBox Detector (SSD) [52], and RetinaNet [45], [53]. The key concept behind this type of architecture is the usage of regressing the predictions for the bounding boxes. Therefore, constructing a bounding box just with few values grants the opportunity for effortless and quicker detection and classification. An illustrative example is provided down below in (Figure 10) [54].

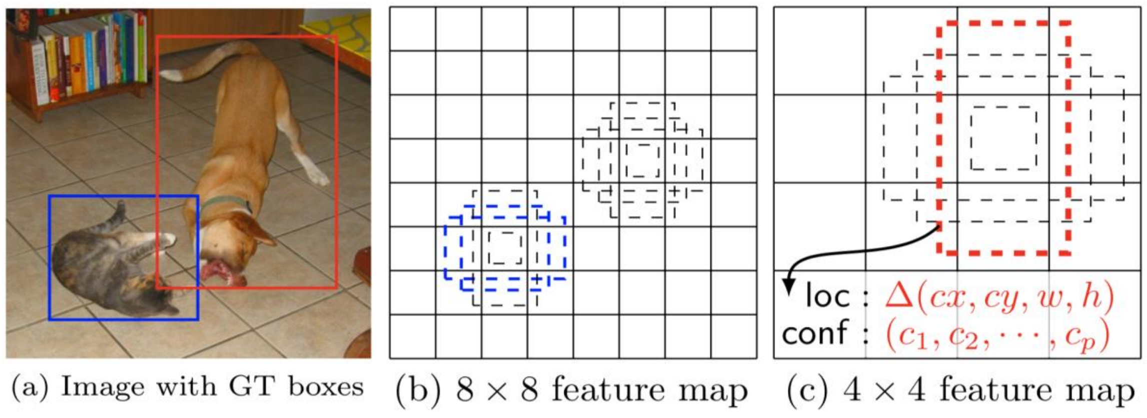


Figure 10: Example of SSD framework. (a) An instance of the training set, which contains a number of images along with the ground truth boxes for each object. (b) A feature map with dimensions 8x8 provides different sets of boxes for the two different locations for the various perspectives. (c) A 4x4 feature map, along with the predicted location and confidence values.

To this point is important to introduce the concept of Real-Time Object Detection. As the name suggests, Real-Time Object Detection is when you apply object detection in real-time, where multiple images per second are given as input to a model. Hence, it is significant to maintain a relatively fast inference without downgrading the accuracy level. Effective models are the most suitable for this incidence, as they can deal with the increased computational power needs and ensure a basic level for accuracy [55].

3.6 Single Shot MultiBox Detector (SSD)

Single Shot MultiBox Detector or SSD is an architecture that belongs to the feed-forward convolutional network methods, which generates several bounding boxes along with the corresponding scores, for the different objects that are being identified, utilizing a non-maximum suppression stage before executing the final detections [54]. SSD was introduced back in 2016 by W.Liu *et al.*, being the second one-stage architecture related to object detection and by extension to the deep learning field [45]. Surprisingly, it was able to achieve remarkable results for both and performance and precision, recording over 74% for mean Average Precision (mAP) at 59 frames per second (FPS) on several datasets including the PASCAL VOC, the COCO, and the ILSVRC dataset. In the experiments conducted on the above cases, SSD outperforms other models just like Fast R-CNN, Faster R-CNN, and YOLO for the great majority of the tested objects.

The basic idea behind the architecture of the SSD is that it takes only one shot in order to identify the various objects contained in an image, based on the multibox feature. Typically, the structure of an SSD model or alternatively the base network, comprised of standard architecture, named VGG-16, a common CNN being utilized to classify the high-quality images. Sequentially, there are multiple additional feature layers with their size being reduced gradually offering recognitions at various scales. Each of the above feature layers produces a different model, providing different predictions. As the image passes through the multiple feature layers a prediction about the bounding boxes along with the class score and the 4 offsets relating to the shape of the bounding boxes, is conducted in every separate stage. The progressively smaller feature layers try to cover larger-scale receptive fields, providing us with more general representations [4], [56]. The structure of the SSD is presented in (Figure 11) [54].

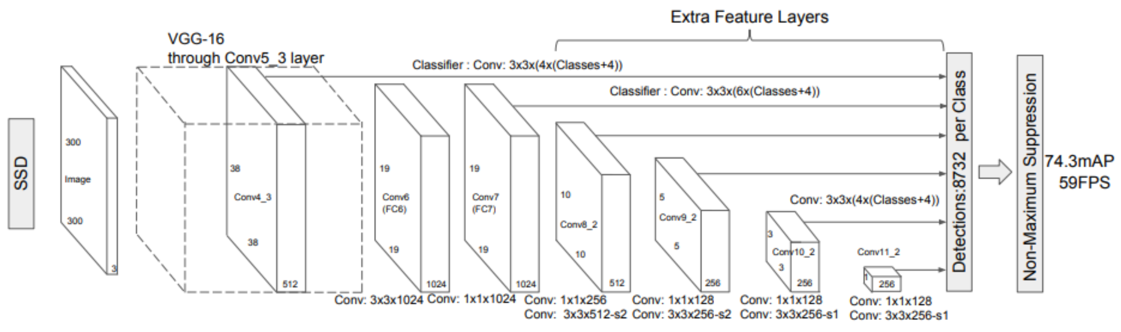
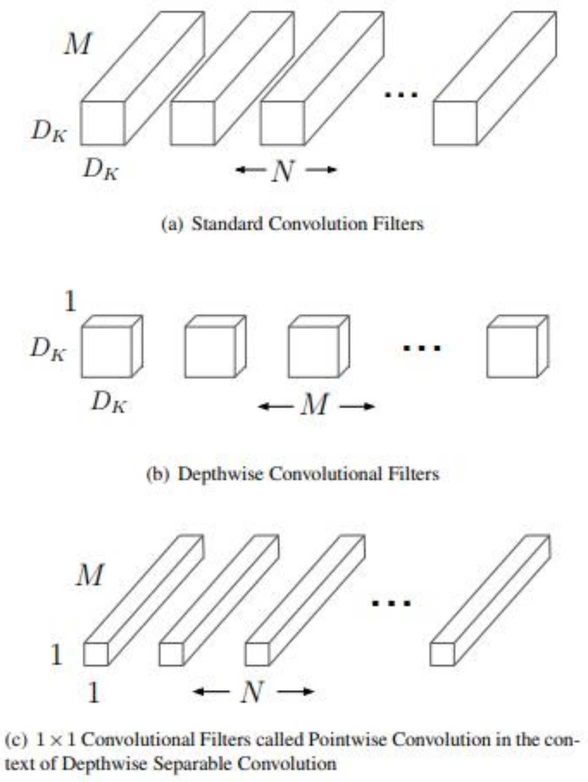


Figure 11: An example of SSD architecture. In the above Figure, it is apparent that VGG-16 is followed by convolutional layers with various sizes decreasing progressively, ending up to a non-maximum suppression filter to eliminate the duplicate predictions and to assort them by the confidence score.

3.7 MobileNets

A family of very skillful models that performs sufficiently well, especially when used for mobile and other embedded devices is that of MobileNets. This particular category of models is based on a streamlined architecture that utilizes depth-wise divisible convolutions in order to construct a lightweight convolutional neural network, offering improved performance and small latency [88]. Simply, by altering the appropriate hyper-parameters and depending on the problem’s constraints it is able to provide an efficient trade-off between the latency and the accuracy of a model. MobileNets modify the ordinary convolutional neural network architecture analyzed previously, by using two new layers, a depth-wise convolution and a point-wise convolution filter instead of the standard convolution one. The former of the two new layers outperforms the standard convolution in terms of effectiveness and is responsible for filtering the inputs. Subsequently, point-wise convolution combines linearly the output of the previous layer, with a 1x1 convolution. The two filters together are well-known as depth-wise separable convolution. The architecture of the described MobileNets is demonstrated in the following (Figure 12).



M
D_K
D_K
$\leftarrow N \rightarrow$
(a) Standard Convolution Filters
1
D_K
D_K
$\leftarrow M \rightarrow$
(b) Depthwise Convolutional Filters
M
1
1
$\leftarrow N \rightarrow$
(c) 1 × 1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 12: Example of MobileNets architecture. (a) Standard convolution filter. (b), (c) Are combined to generate a depth-wise separable convolution.

3.8 Tensorflow

Tensorflow consists of an end-to-end open-source library for both numerical calculations and large-scale machine learning [57]. It is very useful for a great variety of tasks and especially in training and inference for deep learning algorithms. Tensorflow models offer us the ability to utilize them with little or no change for numerous diverse heterogeneous systems, including mobile phones, tablets, and even more complicated cases, just like GPU cards, which are using a plentiful amount of machines and devices [58]. The system was developed in 2015, under the Apache license, by a Google's team, named Google Brain, aimed mainly for internal ML usage and purposes. Tensorflow computations incorporate the notion of the graph, which is comprised of several nodes. This graph is portrayed by a data flow, as we can also observe in the illustrative example in (Figure 13), enabling flexible maintenance and update for each node [59]. Soon this machine learning system became very popular, as the innovative optimizations and the easier and more efficient training made a great impact. Tensorflow models can achieve a forceful performance and are suitable for numerous real-world applications, working at the same time especially well on mobile devices.

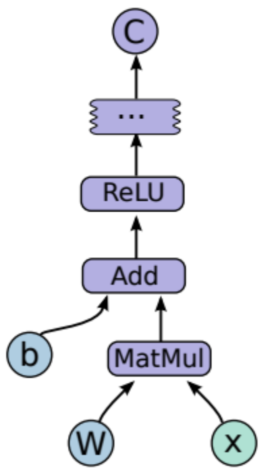


Figure 13: Example of a computation graph.

At this point, it is noteworthy to mention that currently, Tensorflow offers two different methods for developing an application on mobile devices, Tensorflow Mobile and Tensorflow Lite.

3.8.1 Tensorflow Mobile

The former of the two versions, Tensorflow Mobile, is the oldest approach. It is utilized for machine learning purposes for mobile platforms just like Android and iOS. Consequently, a user has the potential to select a new or a pre-trained model, add the appropriate libraries into his application, prepare the created model file and run the necessary optimizations, so as to eventually obtain the desired mobile application. Regularly, such mobile machine learning instances may contain Image Classification, Object Detection, Segmentation, Text Classification, and many more [57].

The continuous data transfer between the mobile devices and the data centers can be characterized as a particularly time-consuming and costly solution. At this point, using mobile machine learning and more specifically Tensorflow Mobile, there is the opportunity for computations to be conducted on the devices instead, saving a considerable amount of network delays and computational power. Such type of tasks often demands a great computational power, translated to hardware requirements involving CPUs, GPUs and bespoke ASICs, famous with the name Tensor Processing Units (TPUs). Notwithstanding, the special architecture of a Tensorflow model, as well as the model optimization can substantially improve the total performance, reducing the required calculations, and providing satisfying results with low latency [58].

3.8.2 Tensorflow Lite

The latter of the two versions, is the Tensorflow Lite, a newer and a lightweight deep learning framework, individually developed for mobile and embedded cases, presenting enhanced performance and smaller application size [60]. In addition, hardware acceleration plays a significant role in the overall performance of the models. Tensorflow Lite employs a compressed file format *.tflite*, founded on an efficient open-source serialization library, named FlatBuffers. Sequentially, after the training stage of the model, the appropriate transformation, and the creation of the *.tlite* file, the file is loaded into a mobile interpreter [61]. The following (Figure 14), represents the structure of the Tensorflow Lite architecture.

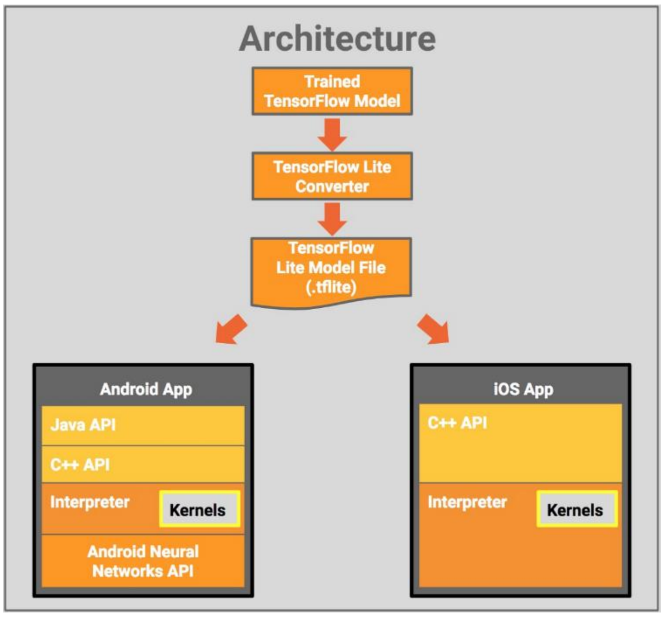


Figure 14: Tensorflow Lite architecture.

3.8.3 Tensorflow Mobile vs Tensorflow Lite

Tensorflow Lite is considered the evolution of the Tensorflow Mobile. Typically, it presents ameliorate results in both performance and binary size of the model, regarding the older Tensorflow Mobile. Nonetheless, inasmuch that Tensorflow Lite is still in a developer preview, Tensorflow Mobile generally constitutes a more stable solution, supporting multiple existed or customized models.

In this thesis, we focus on Tensorflow Lite, as the desired models belong to the streamlined architecture, MobileNets [62], which is among the several operations that are being supported.

3.9 Transfer Learning

A strategy that is commonly utilized to train a model in Machine Learning and by extension in Deep Learning, is that of Transfer Learning. Often, the procedure of learning in a target domain requires valuable knowledge, which may demand a huge amount of time to train from scratch or to gather all the desired data and computational power. Hence, a usual technique to follow is to use pre-trained models, models that are already trained on prior collected, and related training data [63]. Alternatively, the knowledge gained easily from learning on one domain could be also employed to a related problem on a different domain. The above-described technique, where a pre-trained network is used, is referred to as Transfer Learning. The approach of Transfer Learning can be applied to various domains and tasks offering optimization, enabling quick progress, and enhanced performance for the specified task. (Figure 15) represents the difference between the Traditional and the Transfer Learning methods. For the first case, the main operation is to learn each separate task individually and from scratch, in contrast with Transfer Learning where knowledge from previous related instances is transferred allowing the potential for significantly improving the efficiency of the new target task [64].

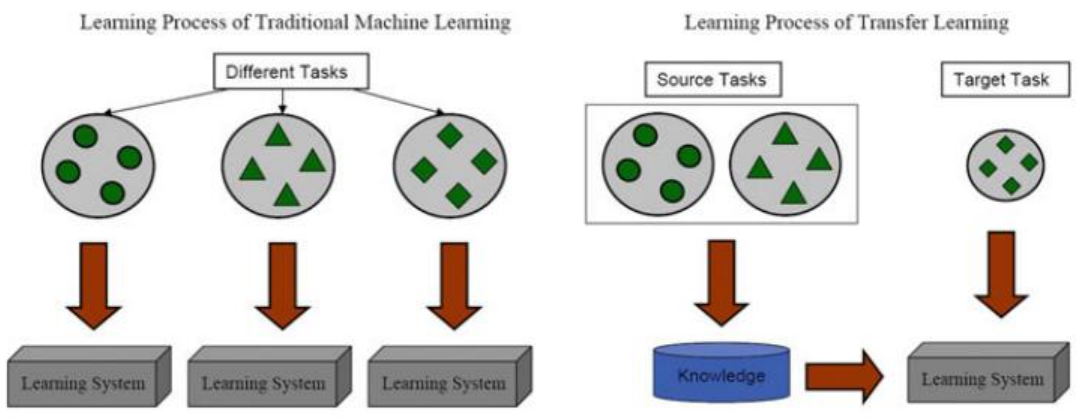


Figure 15: An illustrative example of the difference between the Learning Process in Traditional Machine Learning and in Transfer Learning.

3.10 Evaluation Parameters

Before proceeding with the contribution and the experiments, it is vital to analyze and investigate the evaluation parameters that are usually being utilized in Object Detection. With the term evaluation parameters in Machine Learning, we consider the parameters deployed in order to measure and define the value and the effectiveness of a trained model. In this subchapter, we try to introduce the basic metrics used to assess and evaluate the performance of the SSD models. In other words, we try to comprehend whether a specific model has reached its optimal performance or there is a change for further advancement.

3.10.1 Basic Concepts

At this point, it is significant to introduce some essential concepts, necessary for the better comprehension of other metrics as well. To begin with, the Confidence score is a concept we often may encounter in object detection. It is defined as the probability of an anchor box to contain an object and receives a score between 0 and 100. Simply, a score of 0 is translated as no match has been found, although a score of 100 is likely a perfect match [71]. Therefore, the higher the Confidence score the more significant the confidence for an answer is.

Intersection over Union (IoU) is another common metric in object detection, introduced by Jaccard Index, and measures the overlap between two bounding boxes. A ground truth bounding box Bgt and a predicted Bp bounding box. Depending on the IoU value we can characterize a detection as valid or not. Consequently, as the value of the IoU increases the more accurate the prediction becomes and the greater the chance for an acceptable outcome. Straightforwardly, the IoU receives values between 0 and 1 and can be calculated by the following fraction (3.1),

$$IOU = \frac{area\ (B_p \cap B_{gt})}{area\ (B_p \cup B_{gt})} \tag{3.1}$$

(Figure 15) demonstrates an illustrative example of the IoU between the two different bounding boxes. The ground truth bounding box is depicted with the green color whereas the predicted bounding box with the red one [71].

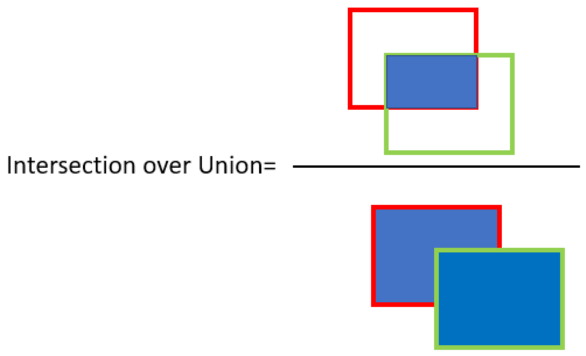


Figure 15: Example of the IoU between the ground truth and the predicted bounding box.

Some also fundamental basic concepts are summarized down below:

- True Positive (TP), when we predict something as positive and actually it is positive, IoU ≥ threshold
- True Negative (TN), when we predict something as negative and actually it is negative (not implemented as it represents all the possible cases where we correctly not detect an object)
- False Positive (FP), when we predict something as positive although in fact it is negative, IoU < threshold
- False Negative (FN), when we predict something as negative although in fact, it is positive (ground truth is not detected) [68], [69]

The threshold may vary determined by the metric and regularly takes values such as 50%, 75%, or 95%. Hence, the higher the value the smaller the chance for a correct prediction.

Moreover, useful and extensively utilized metrics, not only in object detection but generally in machine learning are Precision and Recall. Precision is the metric that gives us an indication of the number of actual positive cases, considering all the positive predictions by our model. The formula for the estimation of the Precision metric is provided down below (3.2),

$$Precision = \frac{TP}{TP+FP} \tag{3.2}$$

Recall is estimated by the formula presented in (3.3) and is the metric that informs us about the number of the positively identified cases by the model, considering all the positive cases,

$$Recall = \frac{TP}{TP+FN} \tag{3.3}$$

Nevertheless, Precision and Recall are two evaluation metrics that are inversely proportional to each other. In other words, when we improve a classifier's precision value, then the recall value is reduced and vice versa. Therefore, many times it is a challenging task to decide between a classifier with better recall score and the other with a better precision. F1-score comes to fill the gap, by combing both performance metrics into a single metric. It can be characterized as the harmonic mean calculated by the following equation (3.4),

$$F1 - score = 2\ x\ \frac{(precision\ x\ recall)}{(precision+recall)} \tag{3.4}$$

The F1-score can receive values between precision and recall but we should also take into consideration that allocates a greater weight to lower values [74].

Now that we have covered some essential concepts just like IoU, TP, TN, FP, FN, Precision, Recall, and F1-score, we can proceed with the explanation of some useful object detection metrics including Average Precision (AP) and mean Average Precision. Before analyzing the aforementioned metrics, it is vital to mention that these evaluation metrics may vary, mainly depending on the particular competition. The most well-known competitions contain the PASCAL VOC Challenge [65], the Common Objects in Context (COCO) Detection Challenge [66], and the Google Open Images Dataset V4 Competition [67].

3.10.2 Average Precision

A way to measure the performance of an object detection model is by calculating the area under the curve, well known also as (AUC), of the Precision x Recall curve (Figure 18). Considering that the Precision x Recall curve often presents a zigzag pattern, constitutes the comparison of the different curves a challenging task. This is where Average Precision comes to help up dealing with the referred issue. As the name suggests, Average Precision is defined as the precision averaged for all the recall values between 0 and 1 [71]. Therefore, it is necessary to locate the area under the Precision x Recall curve.

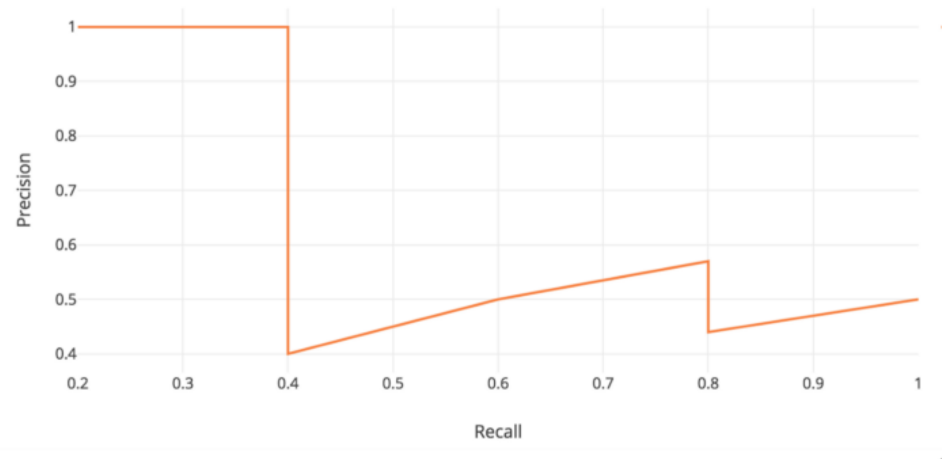


Figure 16: Precision x Recall curve.

PASCAL VOC challenge is very popular as a methodology for computing the Average Precision. It used to be estimated interpolating 11 equally spaced points [72], calculating the average precision among these 11 recall stages [0, 0.1, …]. Simply, the formula for Average Precision with this method is provided down below,

$$AP = \frac{1}{11} \sum_{r \in \{0,0.1,\ldots,1\}} \rho_{interp(r)} \tag{3.5}$$

where

$$\rho_{\text{interp}} = \max_{\tilde{r}:\tilde{r} \geq r} \rho\left(\tilde{r}\right)$$

Nonetheless, currently, the estimation of the Average Precision is based on using the whole amount of the data points, rather than only the 11 interpolating points. Given this circumstance, Average Precision is being computed using the following formula,

$$\sum_{n=0} \left(r_{n+1} - r_n\right) \rho_{\text{interp}}\left(r_{n+1}\right)$$

where

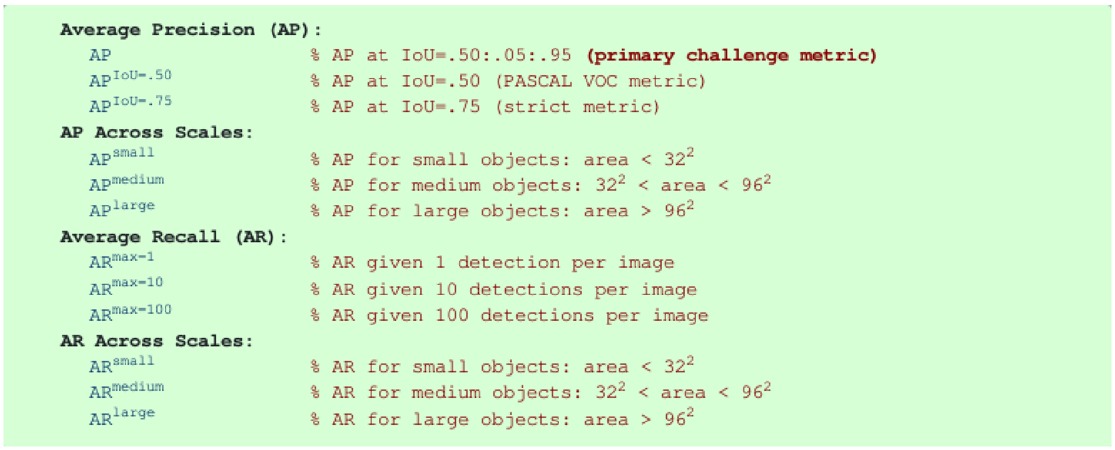
$$\rho_{\text{interp}}\left(r_{n+1}\right) = \max_{\tilde{r}:\tilde{r} \geq r_{n+1}} \rho\left(\tilde{r}\right)$$

3.10.3 Mean Average Precision (mAP)

Another commonly used object detection evaluation metric is that of the mean Average Precision (mAP). Compering both the ground truth and the predicted bounding boxes, the mAP provides a numerical outcome. As the scores receive higher values simply the prediction becomes more accurate. The mAP is the average of Average Precision. Alternatively, in many instances, mAP is computed by averaging the AP for each class. Taking into account that in object detection we have K > 1 classes, the mAP is defined as,

$$mAP = \frac{\sum_{i=1}^{K} APi}{K} \tag{3.6}$$

Notwithstanding, as previously referred, this rule is not applied on every single occasion and different contexts may provide dissimilar results using the same metrics. For instance, in COCO Detection Challenge, the AP and the mAP, which constitutes the primary challenge metric, are the same metric. To clarify this statement, the mAP for the COCO challenge is estimated by taking the mean of the AP of all the classes in association with all the IoU thresholds between 0.5 and 0.95, with step size equal to 0.5. (Figure 19) [73], help us comprehend the 12 different metrics are used to evaluate



Average Precision (AP):	
AP	% AP at IoU=.50:.05:.95 **(primary challenge metric)**
$AP^{IoU=.50}$	% AP at IoU=.50 (PASCAL VOC metric)
$AP^{IoU=.75}$	% AP at IoU=.75 (strict metric)
AP Across Scales:	
AP^{small}	% AP for small objects: area < 32^2
AP^{medium}	% AP for medium objects: 32^2 < area < 96^2
AP^{large}	% AP for large objects: area > 96^2
Average Recall (AR):	
$AR^{max=1}$	% AR given 1 detection per image
$AR^{max=10}$	% AR given 10 detections per image
$AR^{max=100}$	% AR given 100 detections per image
AR Across Scales:	
AR^{small}	% AR for small objects: area < 32^2
AR^{medium}	% AR for medium objects: 32^2 < area < 96^2
AR^{large}	% AR for large objects: area > 96^2

Figure 17: Evaluation metrics used by COCO.

the performance of a model on COCO. We can observe that the mentioned COCO AP deviates from the general standard, where the Average Precision is calculated at a single IoU of 0.50, which is also at the same time the case for the PASCAL VOC challenge. Furthermore, there is the chance for also assessing the detections based on the size of the object (small, medium, large), as much as based on the area they cover (< 32^2, 32^2 < area < 96^2, > 96^2).

4 Contribution and Experiments

In this section, we analyze the procedure and the reasoning behind the conducted experiments, alongside the selection of the specific technique. We aim to implement and evaluate the suitability and the effectiveness of the CNN models for recognizing and detecting landmarks and monuments from various angles and different lighting conditions. As aforementioned in section 1.2, the tested monuments contain the UNESCO World Heritage Monuments of Thessaloniki. The assessed CNN detection models are the "*SSD MobileNet V1 coco*" and the "*SSD MobileNet V2 coco*", as they offer high performance and considerably rapid response, especially advantageous for mobile devices, which involve limited calculative power. In essence, the desired outcome is to provide as accurate predictions for the bounding boxes as possible and in a reasonably small period.

4.1 Preparing the dataset

Probably the most time-consuming part of the experiment consisted of the data gathering and the pre-processing. An essential and inextricable task for every training model in machine learning is the collection of the required dataset (collection of data).

4.1.1 Data Gathering

For the purpose of this thesis, we gathered images regarding the 18 different UNESCO World Heritage Monuments, located in the city of Thessaloniki. Since there was not any existed dataset concerning our aims available, the data collection was created by manually downloaded images from the web, in combination with personal photographs taken for the respective monuments/landmarks. As lighting conditions and different angles of a monument are two factors that directly affect the coloring and possibly the shape of a monument, we try to maintain a variety in our images. With this strategy, it is ensured a greater chance for a correct and accurate prediction, even under various circumstances. In the following example of the White Tower (Figure 18), we can notice how the varied lighting conditions and the different angles of view may provide totally different outcomes. To fulfill our goal, we collected a total amount of 4708 images which are divided into 18 classes. The desired classes are presented in Table 1,

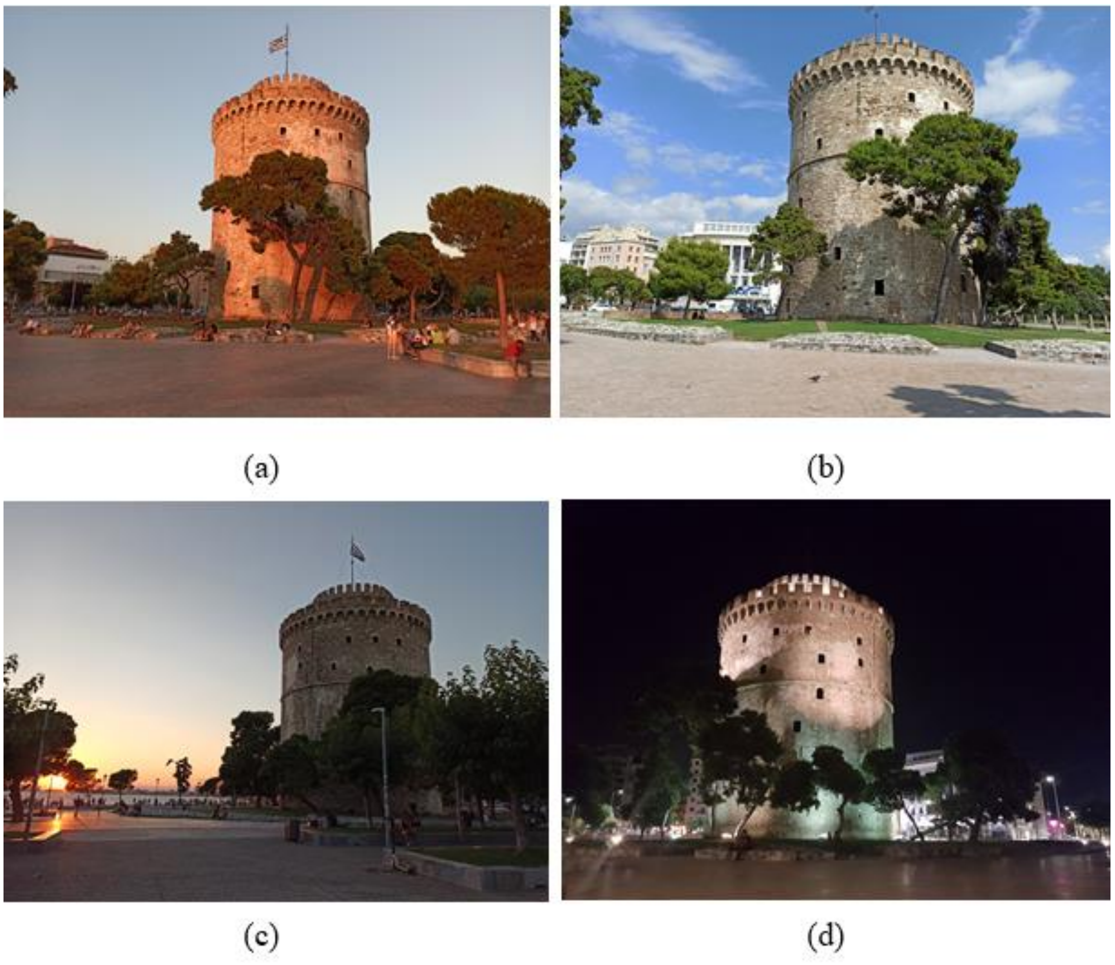


Figure 18: Example of different lighting conditions and angles of view for the White Tower.

alongside the number of images for each monument/landmark.

4.1.2 Data pre-processing

A major step in machine learning is that of pre-processing. In this subsection, we ensure that the collection of the data maintains a standard quality and concurrently we apply the necessary measures to prepare the data for the training section.

Table 1: The number of images provided for each of the UNESCO Monuments of Thessaloniki.

UNESCO Monuments of Thessaloniki	**Monument Image**	**Number of images per monument**
Acheiropoietos		293
Aghios Demetrius		302
Byzantine Baths		95
Heptapyrgion		382
Metamorphosis Sotiros		163
Osios David		100
Panayia Chalkeon		319
Profitis Elias		285
Rotunda		371
Saint Aikaterini		213
Saint Apostoles		249
Saint Nikolaos Orphanos		222
Saint Panteleimon		197
Saint Sophia		329
The Walls		292
Trigonion Tower		274
Vlatadon Monastery		229
White Tower		393

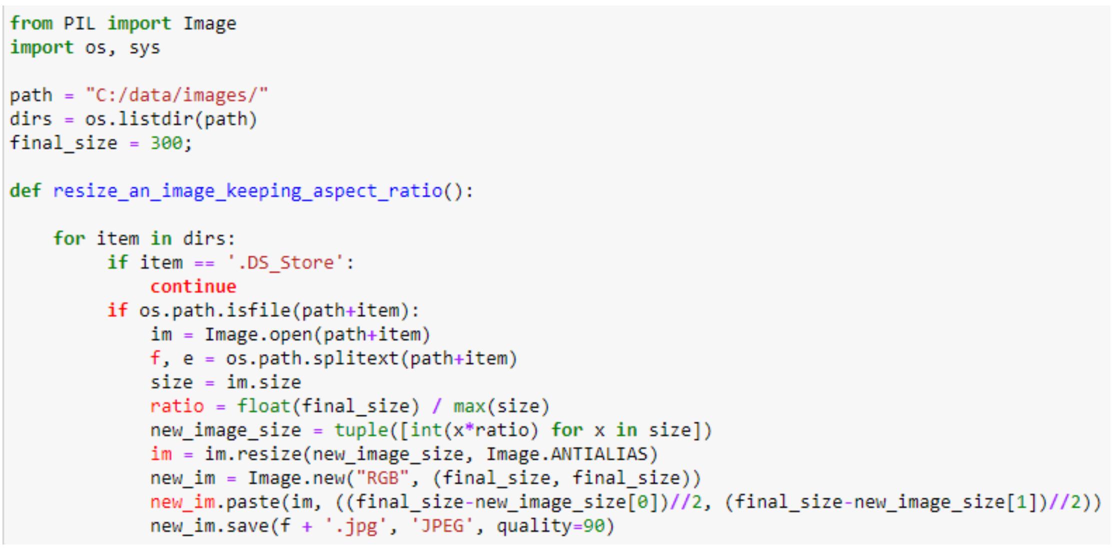
4.1.2.1 Checking the quality

First of all, it is vital to maintain only the images that satisfy a certain resolution. Images with a very low resolution are not useful for our purpose, as the information gained from them is minor and consequently the overall performance is decreasing.

4.1.2.2 Renaming the images

In order to avoid any errors regarding the name of the images, which often poses a great length and contain punctuation marks and spaces, we renamed each image providing a meaningful to us name. Aside from this, we have also the potential to easier pinpoint and process the desirable image.

4.1.2.3 Resizing the images



```
from PIL import Image
import os, sys

path = "C:/data/images/"
dirs = os.listdir(path)
final_size = 300;

def resize_an_image_keeping_aspect_ratio():

    for item in dirs:
         if item == '.DS_Store':
             continue
         if os.path.isfile(path+item):
             im = Image.open(path+item)
             f, e = os.path.splitext(path+item)
             size = im.size
             ratio = float(final_size) / max(size)
             new_image_size = tuple([int(x*ratio) for x in size])
             im = im.resize(new_image_size, Image.ANTIALIAS)
             new_im = Image.new("RGB", (final_size, final_size))
             new_im.paste(im, ((final_size-new_image_size[0])//2, (final_size-new_image_size[1])//2))
             new_im.save(f + '.jpg', 'JPEG', quality=90)
```

Figure 19: Example of the code utilized to resize the images on a specific path.

The resize of the images constitutes a crucial action, especially in case we do not want to overwhelm the model. Furthermore, the reduced resolution will improve dramatically the pre-processing time. Both the SSD_MobileNet_V1_coco and the SSD_MobileNet_V2_coco demand a 300x300 pixels input resolution for the whole amount of images. (Figure 19) demonstrates the method utilized to resize the images located in a specified path, keeping at the same time the original aspect ratio.

4.1.2.4 Annotating the images

A time-consuming but fundamental part of the pre-processing is the annotation of the images. To annotate the images, we use a tool named LabelImg [75], available for several platforms, enabling us an easy draw of the desired bounding box along with the annotation for each box, as seen in (Figure 20). LabelImg also offers us the opportunity for saving the annotation with either YOLO or PascalVOC format. For

our aims we selected the latter of the two formats, which is commonly utilized in

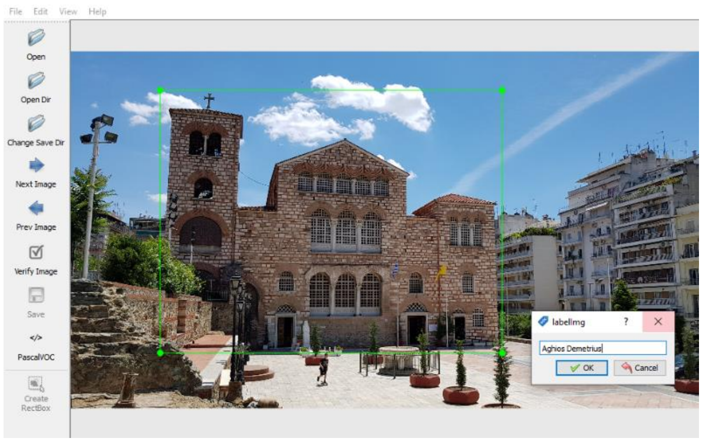
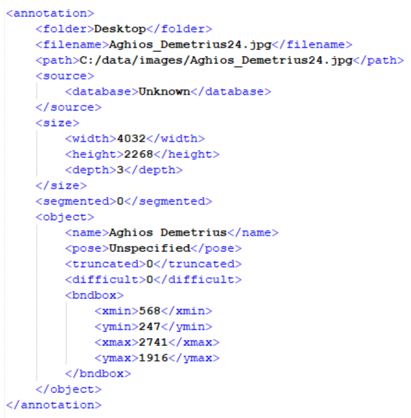


Figure 20: Example of the Labelimg environment. Creating a bounding box and provide an annotation for it.



```
<annotation>
    <folder>Desktop</folder>
    <filename>Aghios_Demetrius24.jpg</filename>
    <path>C:/data/images/Aghios_Demetrius24.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>4032</width>
        <height>2268</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>Aghios Demetrius</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>568</xmin>
            <ymin>247</ymin>
            <xmax>2741</xmax>
            <ymax>1916</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 21: The (.xml) file created after the annotation.

object detection. After the confirmation for the bounding box and the annotation, the program automatically generates a (XML) file with the same name as the image, as presented in (Figure 21). For each image, we can observe that the (XML) file, besides the filename and the size of the image also contains the label name in association with the coordinates of the drawn bounding box/boxes.

4.1.2.5 Splitting the dataset into train and test set

Before proceeding with the next step, we separate our images and their matching annotation files into a training and a testing set, using an 80-20 ratio. The separation is implemented randomly to help ensure homogeneity, allocating the data into two distinct folders, one created for the train and the other for the test set.

4.2 Google Colaboratory

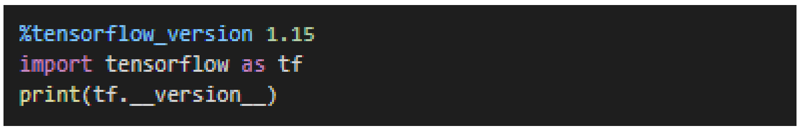
A determining factor for the execution of the project is the selection and the setup of a proper environment. For the implementation and the fulfillment of this thesis, we selected Google’s Colaboratory python environment to perform our experiment and Google Drive [78] to host the necessary data and appropriate tools.

In short, Google Colaboratory [76] or commonly referred to as Google Colab is a research project, offering the potential for students, data scientists, or AI researchers to utilize powerful hardware just like GPUs and TPUs to implement and run their machine learning problems. In addition, Google Colab is based on an interactive Jupyter Notebook framework, equipped with various features [77]. Another main advantage of utilizing its environment is that it has the most frequent libraries for machine learning pre-installed not to mention the fact that also allows us to upload our files and mount our Google Drive.

4.3 Setting up the environment

Another yet important task before training the model is the preparation of the environment, in order to host our data just as the appropriate parameters to conduct the experiment. As mentioned in the previous subsection, we selected Google Drive and Google Colab to help us actualize our goal.

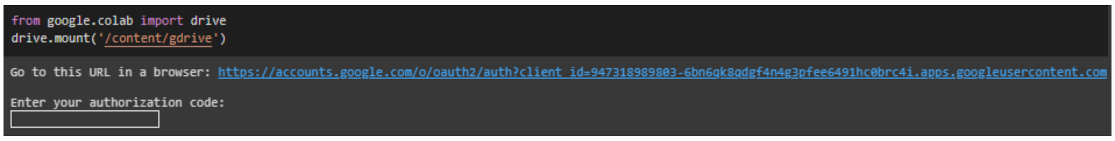
To generate an accurate machine learning model, able to identify the provided monuments/landmarks we used the Tensorflow python library. Although at this time a newer version of Tensorflow has been released, Tensorflow 1.15 (Figure 22) was chosen in our case, as it constitutes a more stable and robust solution regarding the newer one which presents some bugs and is still under development. On the official page of Tensorflow on Github [79], we can find all the necessary files to train a model,



```
%tensorflow_version 1.15
import tensorflow as tf
print(tf.__version__)
```

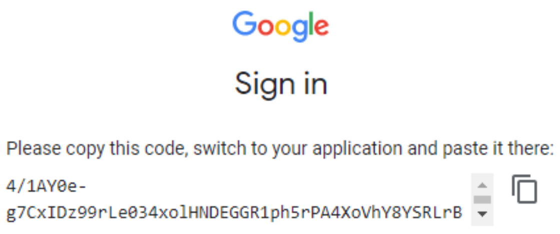
Figure 22: Importing Tensorflow 1.15.

as it is further analyzed down below. In order to access these files, we opted to clone the Tensorflow Github’s repository to our Google Drive account. To achieve this operation, we first mounted a Google Drive account to Google Colab with the following command (Figure 23) and by allowing access inserting the required authorization code (Figure 24). Additionally, now that we have granted access to our account, with the



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

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com

Enter your authorization code:
```

Figure 23: Accessing the Google Drive account.



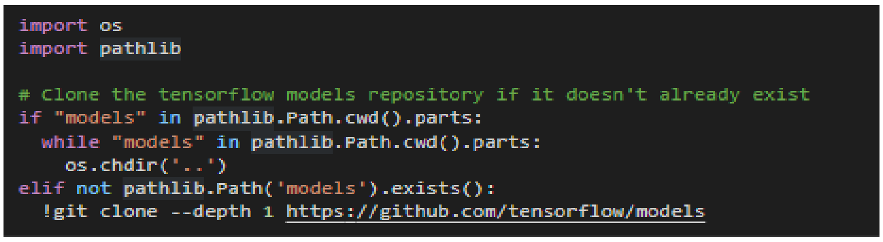
Sign in

Please copy this code, switch to your application and paste it there:

4/1AY0e-
g7CxIDz99rLe034xolHNDEGGR1ph5rPA4XoVhY8YSRLrB

Figure 24: Copy-pasting the provided code, will allow access to our Google Drive repository.

following piece of code (Figure 25), we can clone the specified repository. By changing the working directory, we can start installing the appropriate dependencies that are essential for the experiment.



```
import os
import pathlib

# Clone the tensorflow models repository if it doesn't already exist
if "models" in pathlib.Path.cwd().parts:
  while "models" in pathlib.Path.cwd().parts:
    os.chdir('..')
elif not pathlib.Path('models').exists():
  !git clone --depth 1 https://github.com/tensorflow/models
```

Figure 25: Cloning the Tensorflow Github's repository to our working environment.

For the implementation of the project, we also constructed the proper working directory. More specifically, we created:

- A "deploy" folder, which contains the pre-trained model, the "labelmap.pbtxt" and the "pipeline_file.config", which are being covered in the next subsections.

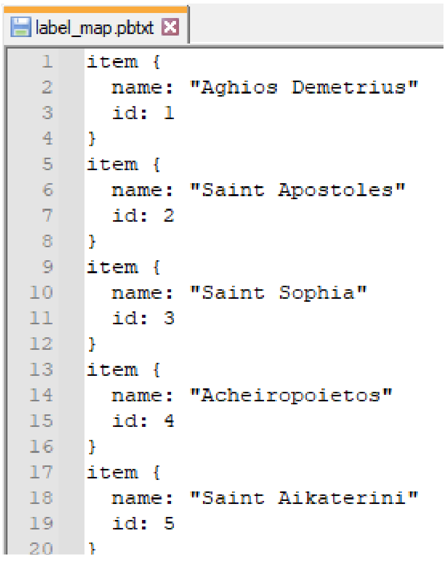
- An “images” folder, which includes the images and the annotations (.xml) files for both the train and the evaluation phase.
- A “data” folder, required to save some additional files regarding the training process.
- An empty “training” folder, to store the checkpoint files of the training procedure.
- An “exported_model” folder, to save the exported model.

4.4 Converting xml files to tfrecord

Another necessary operation is the conversion of the aforementioned (XML) files, which contain the total amount of the annotations, into (TFRECORD) files. Tfrecord is a type of data supported by Tensorflow and is highly recommended, as it offers an effective way of sustaining a scalable architecture and a common input format. The described process occurred in two sequential steps. We first transformed the (xml) files into (CSV) files following by the transformation of the (CSV) files into (TFRECORD) files.

4.5 Creating a label map

As a trained model can only identify an integer value, it is essential to create a label map file. Alternatively stated, we have to map each of our class labels into an integer number. Since our project includes 18 monuments/landmarks, the process should be applied to all 18 classes. Starting with the integer “1”, a sample of the generated label map is illustrated in (Figure 26) and is saved with a (PBTXT) format.



```
item {
  name: "Aghios Demetrius"
  id: 1
}
item {
  name: "Saint Apostoles"
  id: 2
}
item {
  name: "Saint Sophia"
  id: 3
}
item {
  name: "Acheiropoietos"
  id: 4
}
item {
  name: "Saint Aikaterini"
  id: 5
}
```

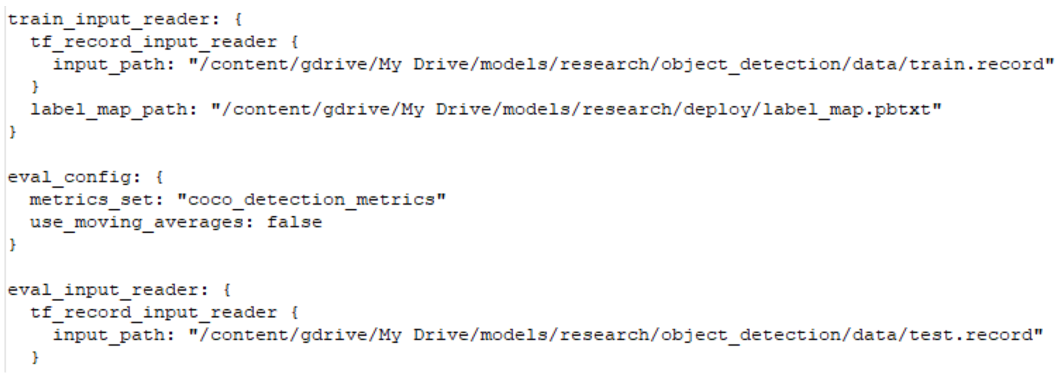
Figure 26: Example of the generated label map.

4.6 Download a pre-trained model and alter the configuration file

Considering that training a complete convolutional neural network from scratch requires significant amounts of time, storing capacity, and computational power, using a pre-trained model is strongly suggested. The idea behind the concept is transfer learning, as referred to at 3.8 and it is being actualized with the Tensorflow Object Detection API [80]. From the Model Zoo repository [81], we can observe various pre-trained models along with their speed and mAP performance pre-trained and tested on the COCO dataset [82]. To conduct our experiment, the “SSD_MobilNet_V1_coco” and the “SSD_MobileNet_V2_coco” models were selected, in view of the fact that both present pleasant performance and great speed, particularly important for mobile devices.

The last step before running our custom object detection model is to download and import the appropriate configuration file for this pre-trained model. As the name suggests, the configuration file defines the exact way of the training process and allows us to modify the desired parameters. The required changes we applied are summarized down below:

- The “num_classes” are defined as equal to 18.
- A “batch_size” of 24 is used.
- The “fine_tune_checkpoint” was set each time regarding the model’s checkpoints name.
- The “input_path” and the “label_map_path” for both “train_input_reader” and “eval_input_reader” are linked with the “train.record”, the “test.record” and the “label_map.pbtxt” files, as seen in (Figure 27).

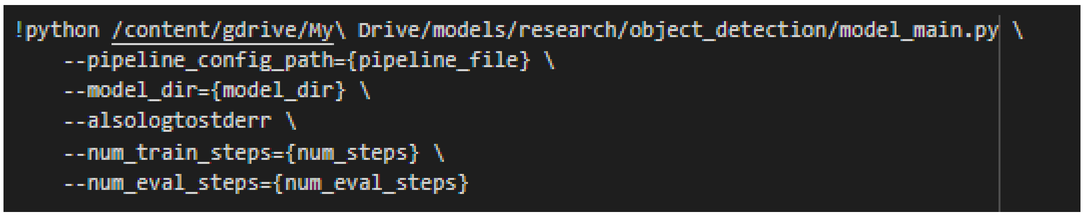


```
train_input_reader: {
  tf_record_input_reader {
    input_path: "/content/gdrive/My Drive/models/research/object_detection/data/train.record"
  }
  label_map_path: "/content/gdrive/My Drive/models/research/deploy/label_map.pbtxt"
}

eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "/content/gdrive/My Drive/models/research/object_detection/data/test.record"
  }
```

Figure 27: A snippet of the configuration file.

4.7 Training the model

It is highly recommended, before start training the model to alter the runtime type to GPU, as Google Colab offering the opportunity to utilize its GPU, which without any doubt can provide significantly faster results. For the training process, the “model_main.py” script from the “object_detection” folder is used, along with several necessary arguments as demonstrated in (Figure 28). More specifically, the



```
!python /content/gdrive/My\ Drive/models/research/object_detection/model_main.py \
    --pipeline_config_path={pipeline_file} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --num_eval_steps={num_eval_steps}
```

Figure 28: Example of the code used for the training process.

following prerequisites are provided:

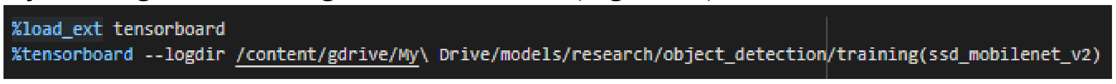
- The path of the “pipeline_file”, which is identical to the configuration file created at the previous step.
- The “model_dir”, which is the path to the output directory, is needed to store the final checkpoint files.
- The “num_steps”, which entails the total number of iterations that the experiment will be conducted.
- The “num_eval_steps”, which defines the number of steps before applying an evaluation.

In the training phase, the model displays the number of steps with their corresponding loss value. Typically, is more preferably a loss value around 1 or less. In addition, it visualizes accumulating evaluation metrics, depending on the progress made until then, and saves the newly learned weights into the specified Google Drive directory. These learned weights are well known as checkpoint files and the more steps we cover the more checkpoints are generated. For every newly trained weight learned, three different types of checkpoints are created. The “model.ckpt-xxx.data” stores the value of each variable, the “model.ckpt-xxx.info” contains metadata for tensors and the “model.ckpt-xxx.meta” saves the graph structure of the model.

The summarized evaluation metrics may vary and are based on the “metric_set” used in the evaluation block of the configuration file, as shown in (Figure 27). The supported evaluation protocols include three different protocols and can be enabled by inserting the corresponding method [83]. When the total amount of steps is covered or we manually interrupt the process the training stops and the saved checkpoints until that point are available for manipulation.

4.8 Measuring and evaluating the model with Tensorboard

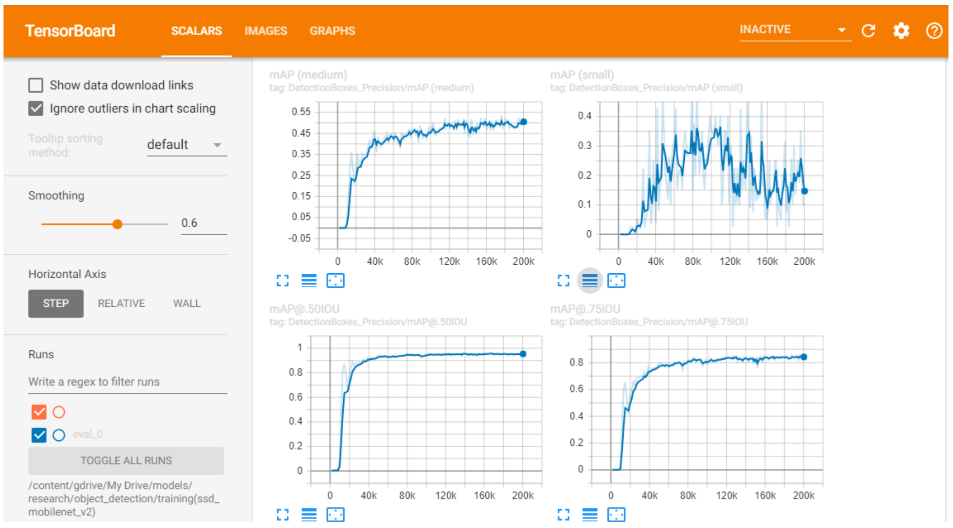
This is an optional action and it does not affect the remaining procedure, however, usually, it is practical and essential to evaluate the model's performance, so as to have a clearer view. Tensorboard is a tool designed to provide various measurements and visualizations, regarding the trained model, just like the loss and the accuracy values. By running the following block of code in (Figure 29), the Tensorboard extension is



```
%load_ext tensorboard
%tensorboard --logdir /content/gdrive/My\ Drive/models/research/object_detection/training(ssd_mobilenet_v2)
```

Figure 29: Loading Tensorboard by providing the training directory of the model.

loaded and provides us with various metrics. Regularly, these metrics are based on the "metric_set" chosen in the configuration file and each selection may offer us a different kind of evaluation. An illustrative example of Tensorboard is presented in (Figure 30), providing us with several graphs regarding the mAP values for different cases as the



Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing: 0.6
Horizontal Axis: STEP
Runs
/content/gdrive/My Drive/models/research/object_detection/training(ssd_mobilenet_v2)

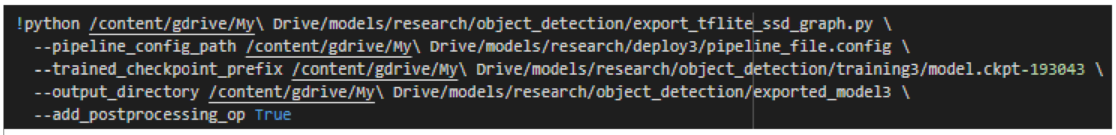
TensorBoard
SCALARS
IMAGES
GRAPHS
INACTIVE
RELATIVE
WALL
Write a regex to filter runs
eval_0
TOGGLE ALL RUNS
mAP (medium)
tag: DetectionBoxes_Precision/mAP (medium)
mAP (small)
tag: DetectionBoxes_Precision/mAP (small)
mAP@.50IOU
tag: DetectionBoxes_Precision/mAP@.50IOU
mAP@.75IOU
tag: DetectionBoxes_Precision/mAP@.75IOU
0
40k
80k
120k
160k
200k

Figure 30: Example of the Tensorboard interface.

number of steps increases. Apart from the scalars, Tensorboard may also give us information concerning the structure of the model, the distribution of the weights, the images in case of object detection, comparing the ground truth with the predicted and many others [84].

4.9 Exporting the graph from the trained model

In this step, the “export_tflite_ssd_graph.py” script is used (Figure 31) to generate a frozen graph from the created checkpoint files. It constitutes a fundamental procedure



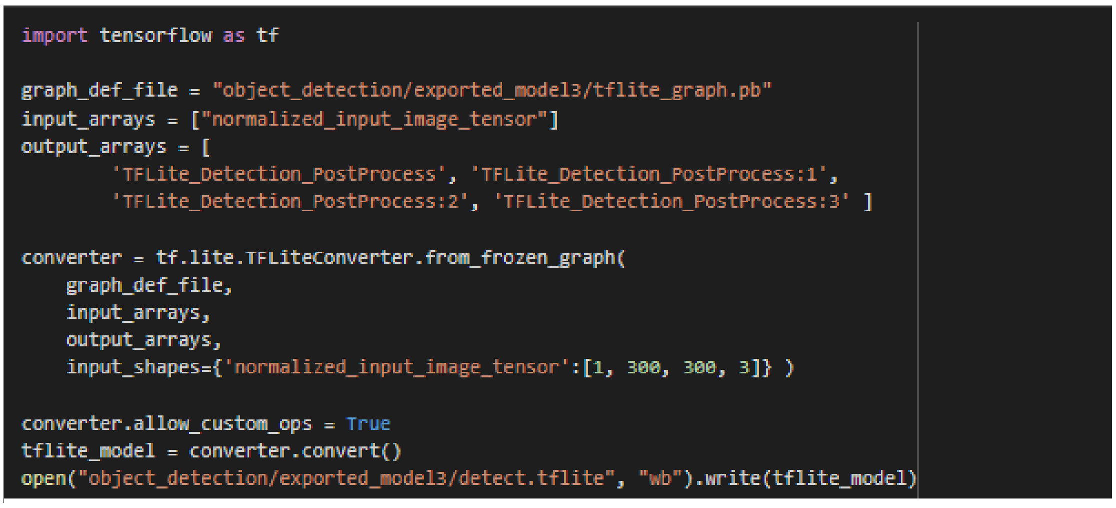
```
!python /content/gdrive/My\ Drive/models/research/object_detection/export_tflite_ssd_graph.py \
  --pipeline_config_path /content/gdrive/My\ Drive/models/research/deploy3/pipeline_file.config \
  --trained_checkpoint_prefix /content/gdrive/My\ Drive/models/research/object_detection/training3/model.ckpt-193043 \
  --output_directory /content/gdrive/My\ Drive/models/research/object_detection/exported_model3 \
  --add_postprocessing_op True
```

Figure 31: Example of exporting a Tensorflow frozen graph.

as transforms the produced files from the training section into a usable format for testing and deploying our model in Android. To execute the above command, we should also specify the path for the configuration file, the desired step checkpoint just like the path we would like to store the exported model. A common approach concerning the selection of the appropriate step checkpoint is to take the latest one, as it has covered the most steps. Nonetheless, this may not be the case for any occasion and for this reason, it is recommended to first consult an evaluation metric. If the process of exporting the graph is properly conducted, then two new files will be created at the specified directory, a “tflite_graph.pb” and a “tflite_graph.pbtxt”.

4.10 Getting the model’s lite version with TFLite

In order to deploy our model in the Android platform, it is necessary first to convert the created “tflite_graph.pb” file into a Tensorflow Lite format. Using a special format for manipulating models, Tensorflow Lite offers efficient execution for mobile and other embedded devices, utilizing limited computational power. To perform the conversion, the Tensorflow Lite converter tool was used [85]. There are two options for conversion, either to generate a Non-quantized model or a quantized one. The former of the two presents slightly better results and requires more storing space, although the latter of the two provides slightly worst results requiring fewer storing space. For our purpose, the Non-quantized approach was selected (Figure 32), as the better performance was more significant and the extra storing space was inconsiderable. After the correct execution, a (TFLITE) file is created.



```
import tensorflow as tf

graph_def_file = "object_detection/exported_model3/tflite_graph.pb"
input_arrays = ["normalized_input_image_tensor"]
output_arrays = [
        'TFLite_Detection_PostProcess', 'TFLite_Detection_PostProcess:1',
        'TFLite_Detection_PostProcess:2', 'TFLite_Detection_PostProcess:3' ]

converter = tf.lite.TFLiteConverter.from_frozen_graph(
    graph_def_file,
    input_arrays,
    output_arrays,
    input_shapes={'normalized_input_image_tensor':[1, 300, 300, 3]} )

converter.allow_custom_ops = True
tflite_model = converter.convert()
open("object_detection/exported_model3/detect.tflite", "wb").write(tflite_model)
```

Figure 32: Example of converting the model to Tensorflow Lite format, using the non-quantized method.

4.11 Adding a metadata file to the transformed model

The final step before the implementation of the model into Android is the creation of a metadata file. Alternatively, a file that provides knowledge about the model which is comprised of both human and machine-readable segments. This file is divided into three subcategories, model, input, and output information. After executing the appropriate commands, the metadata file is attached to the defined model. Optionally, with the MetadataDisplayer tool, we have the potential to visualize the results, writing the metadata file into a (JSON) file.

4.12 Deploying the model to Android

In this section, Android Studio 4.1 [86] is used to create and build an application based on our already trained custom model and regarding the aforementioned monuments/landmarks. Briefly, Android Studio grants an integrated development environment for the Android platform, especially constructed for Android development [87]. In order to deploy our custom model, the following procedure was applied:

- Tensorflow's Lite object detection sample repository was cloned into our local repository.
- Initializing the Android Studio application, "Open an existing Android Studio project" was selected. Subsequently, navigating through our repository, the cloned Android directory is chosen (Figure 33).

- If requested, we confirm to apply Gradle Sync and we proceed with the installation of various platforms and tools.
- At this point, we search for the asset folder, located in the "Android/app/src/main" directory and we need to change these files with our

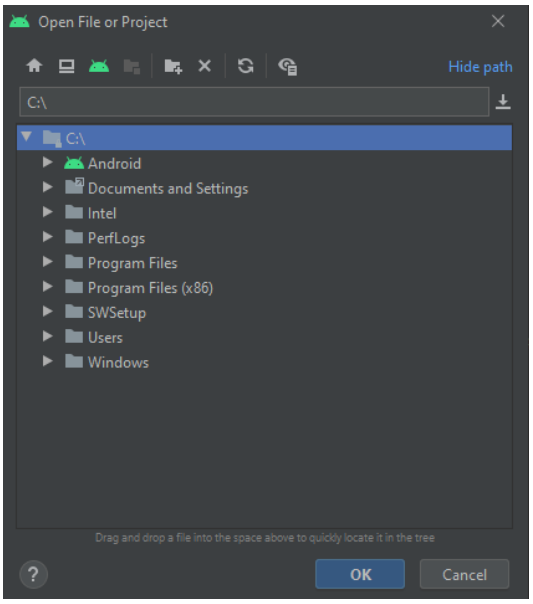
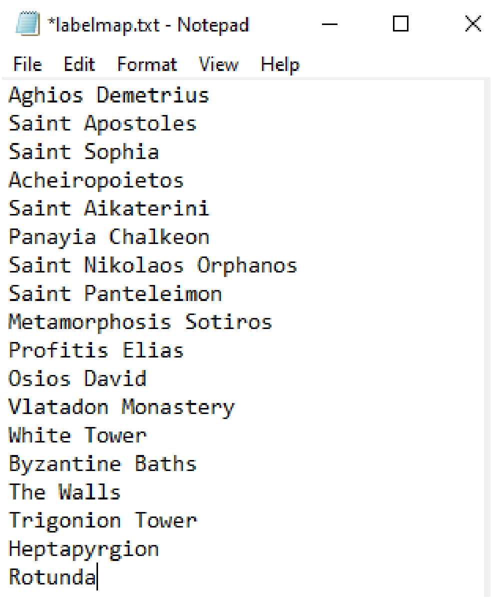


Figure 33: Opening an existing project on Android Studio and selecting Android.

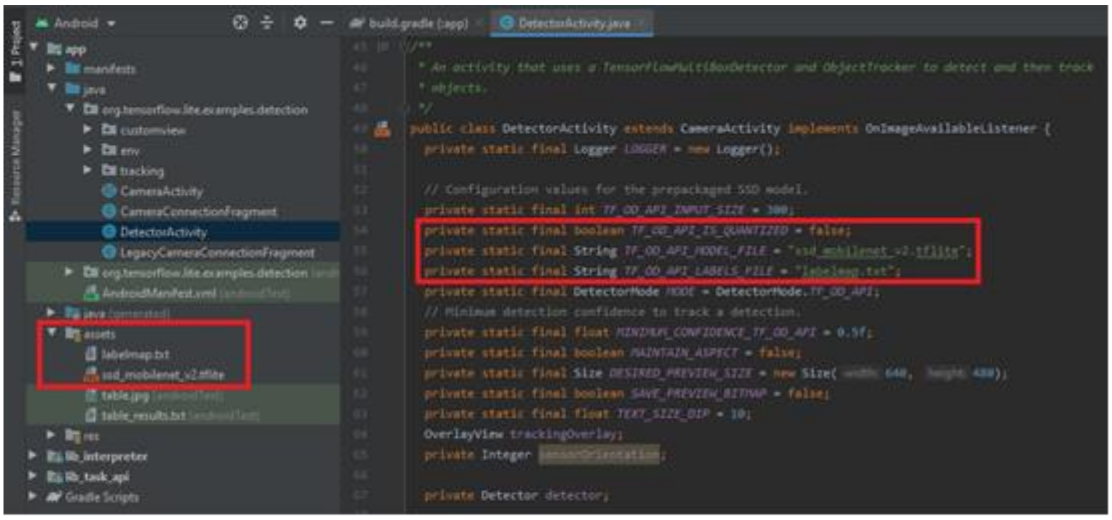


```
Aghios Demetrius
Saint Apostoles
Saint Sophia
Acheiropoietos
Saint Aikaterini
Panayia Chalkeon
Saint Nikolaos Orphanos
Saint Panteleimon
Metamorphosis Sotiros
Profitis Elias
Osios David
Vlatadon Monastery
White Tower
Byzantine Baths
The Walls
Trigonion Tower
Heptapyrgion
Rotunda
```

Figure 34: The structure of the labemap.txt file.

personal files. These are consist of the "ssd_mobilenet_v2.tflite" which is our trained model and the "labelmap.txt" which contains the labels of our trained classes, as shown in (Figure 34).

- Several alterations should also be applied at the "DetectorActivity.java" file, so as to deploy our custom model. We change the "*TF_OD_API_MODEL_FILE*" into our model's name. The same procedure is followed for the "*TF_OD_API_LABELS_FILE*" variable, providing the name of our label map file. Additionally, we modify the "*TF_OD_API_IS_QUANTIZED*" value to "false", considering that our model is a Non-quantized one. (Figure 35) provides an illustrative example of the “DetectorActivity.java” after the above implementations.



```
/**
 * An activity that uses a TensorFlowMultiBoxDetector and ObjectTracker to detect and then track
 * objects.
 */
public class DetectorActivity extends CameraActivity implements OnImageAvailableListener {
  private static final Logger LOGGER = new Logger();

  // Configuration values for the prepackaged SSD model.
  private static final int TF_OD_API_INPUT_SIZE = 300;
  private static final boolean TF_OD_API_IS_QUANTIZED = false;
  private static final String TF_OD_API_MODEL_FILE = "ssd_mobilenet_v2.tflite";
  private static final String TF_OD_API_LABELS_FILE = "labelmap.txt";
  private static final DetectorMode MODE = DetectorMode.TF_OD_API;
  // Minimum detection confidence to track a detection.
  private static final float MINIMUM_CONFIDENCE_TF_OD_API = 0.5f;
  private static final boolean MAINTAIN_ASPECT = false;
  private static final Size DESIRED_PREVIEW_SIZE = new Size( width: 640,  height: 480);
  private static final boolean SAVE_PREVIEW_BITMAP = false;
  private static final float TEXT_SIZE_DIP = 10;
  OverlayView trackingOverlay;
  private Integer sensorOrientation;

  private Detector detector;
```

Figure 34: Example of the “DetectorActivity.java”, after the applied alterations.

- At the “build.gradle” file, we also select the appropriate “complileSdVersion” and the desired “targetSdkVersion” and we deactivate the auto-download of the model.
- Finally, by connecting our Android device to our personal computer, with the developer mode enabled and by selecting “Make a project” our application is ready for use. Alternatively, the Android Studio environment allows us to

create a virtual device offering the opportunity for various tests, possibly at different Android versions.

Figures 36-37, represent an example of the produced application, as well as the user's environment and the first predictions conducted toward the monuments/landmarks.

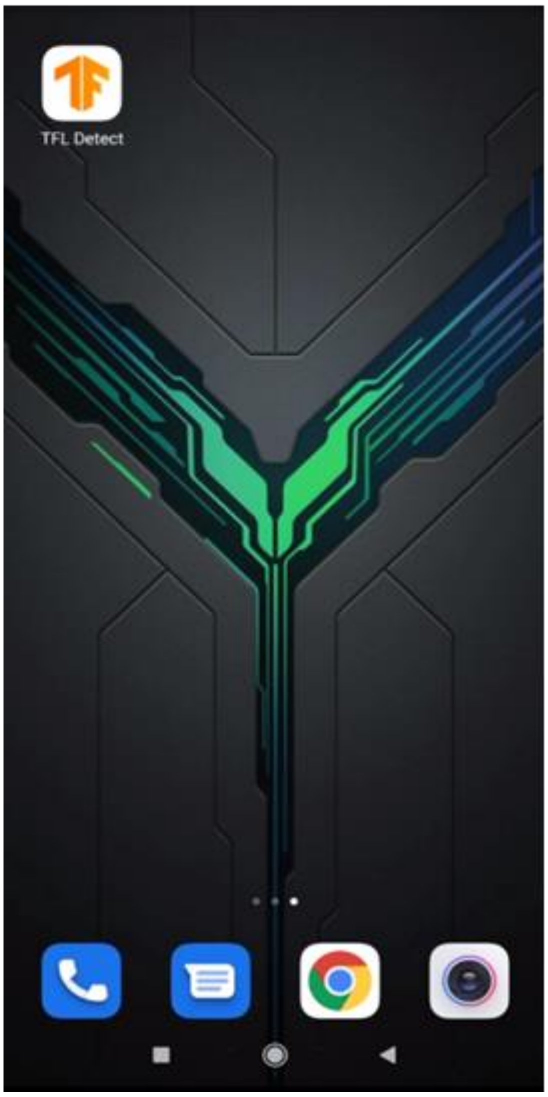
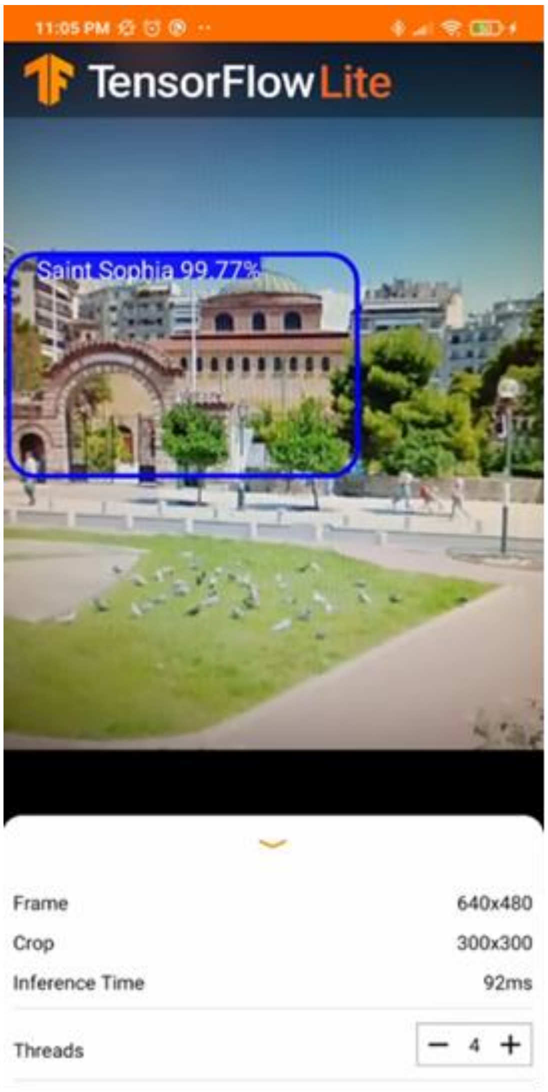


Figure 36: Example of the created application, "TFL Detect".



Frame 640x480
Crop 300x300
Inference Time 92ms
Threads 4

Figure 37: Example of the user's application environment. The picture displays a correct identification of the Church of Saint Sophia in the area of Thessaloniki, along with a bounding box and a confidence score.

5 Results

The object of this section is to investigate whether deep convolutional neural networks are able of providing quick and accurate predictions for various monuments and landmarks and more specifically, whether implementing them on mobile devices, which by definition have limited capabilities, may also present satisfying results. To examine the above statement, two Tensorflow models, the ssd_mobilenet_v1 and the ssd_mobilenet_v2, were trained and evaluated. For the experiment, 4708 images for the 18 different monuments were utilized to train both of the models, performing 200,000 steps each. Moreover, for the training process, an IoU threshold of 50% was selected for defining a prediction as valid or not.

5.1 Mean Average Precision Performance

In order to pinpoint the optimal checkpoint step that maximizes the performance of the models for recognizing the monuments, first the models were assessed in terms of mAP performance. The performance for the two models is demonstrated in Table 2-3, displaying the fluctuation for the mAP for three different percentages of IoU. The computation of the mAP was executed as described in chapter 3.9.3. As the transfer learning technique was implemented in both cases, the two models started learning from

Table 2: Ssd_mobilenet_v1 mAP performance

Step	**mAP**	**mAP_0.50_IOU**	**mAP_0.75_IOU**
1.622	1.189E-02%	9.358E-02%	3.67E-06%
24872	53.88%	85.68%	59.53%
68146	64.51%	92.31%	76.48%
89725	67.36%	93.35%	78.30%
126067	70.29%	94.64%	81.83%
163119	67.87%	93.32%	77.60%
199986	71.90%	95.05%	82.94%
210000	68.96%	93.66%	77.83%

Table 3: Ssd_mobilenet_v2 mAP performance

Steps	**mAP**	**mAP_0.50_IOU**	**mAP_0.75_IOU**
1712	3.950E-02%	3.178E-01%	1.04E-04%
31621	60.73%	88.73%	69.71%
65954	68.63%	93.96%	80.84%
102733	72.00%	95.63%	83.48%
137659	72.70%	96.20%	83.73%
165487	73.59%	95.73%	84.50%
197470	73.32%	95.03%	85.03%

standard pre-trained weights, presenting poor initial values. Nonetheless, within the next 20-30k steps, they rabidly elevated reaching a fixed level, from where and then they exhibited only small deviations. After this stage of exponential boost, both models increased their performance progressively with a reduced growth rate, until they reached a peak. The peak point is different for each case. The ssd_mobilenet_v1 presented its best mAP performance of 71.90% at 199986 steps, whereas the ssd_mobilenet_v2 displayed a max mAP of 73.59%, earlier at 165487 steps. Since the first of the two models demonstrated its best results at the last defined step, it tested for additional 10k steps and it was found out that the values for all of the three metrics lessened.

Furthermore, concerning that the decided threshold percentage is equal to 50% which appertains to the PASCAL VOC metric (AP with IoU .50), this metric reflects to the correctly classified images and at the same time valid predictions (IoU >= .50), as illustrated in (Figures 38-39). The following figures provide a clearer view of the described model's behavior, verifying that they acquired the majority of their knowledge just at the first 20k-40k steps, followed by a steady course until they cover the predetermined number of steps. For both of the models, a mAP (.50 IOU) of more than 95% has been recorded.

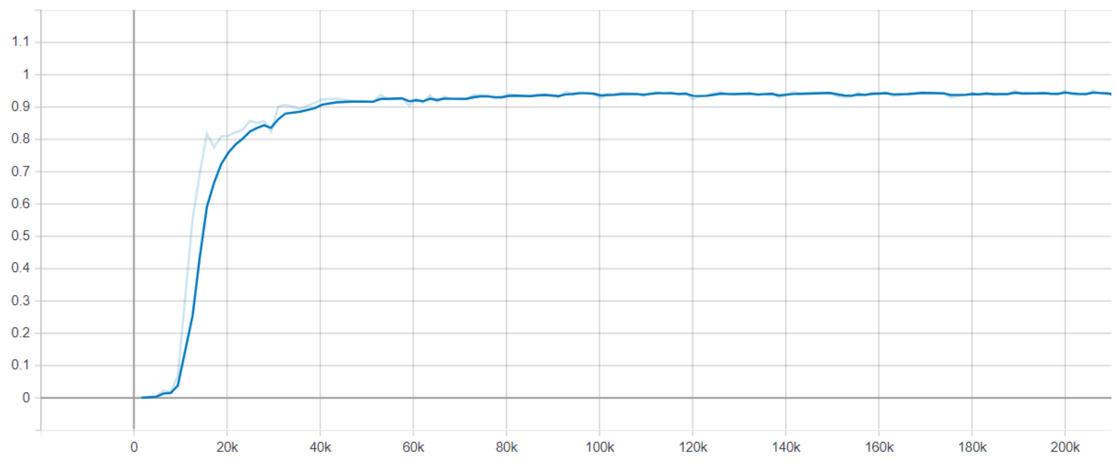


Figure 38: The mAP (.50 IOU) of the ssd_mobilenet_v1 model.

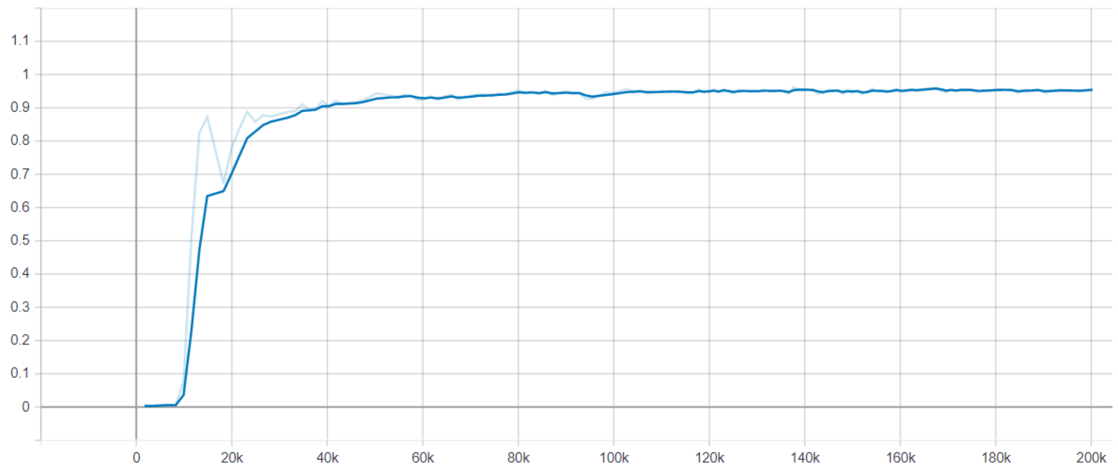


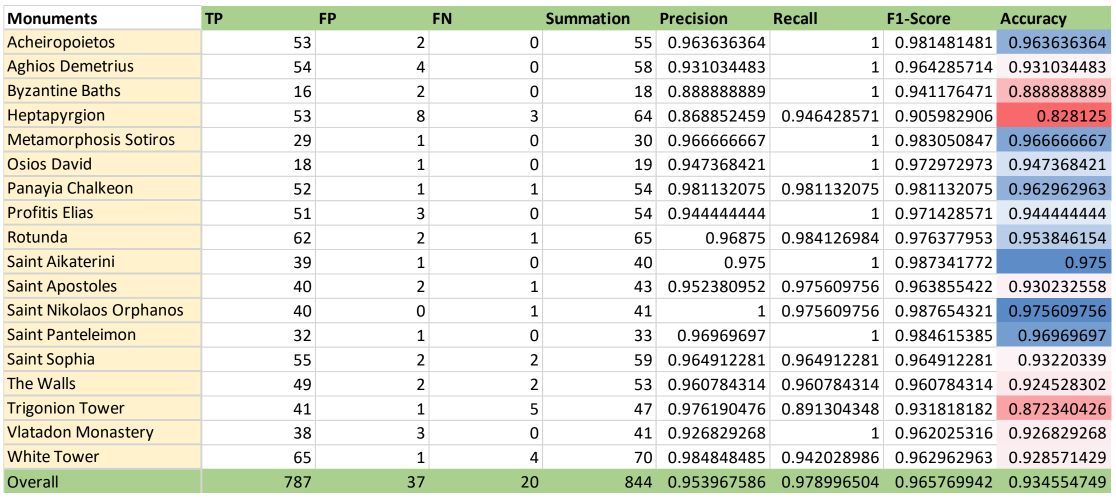
Figure 39: The mAP (.50 IOU) of the ssd_mobilenet_v2 model.

5.2 Additional Evaluation Metrics

As also mentioned in chapter 4.9, by selecting the specific checkpoint steps for the two distinct occurrences and running the appropriate block of code, a frozen graph for each model has been generated. In order to acquire a more rounded overview of the model's performance, each of these two graphs was tested independently on 844 images from all the 18 monuments and assessed in terms of TP, FP, FN, Precision, Recall, F1-Score, and Accuracy.

5.2.1 SSD MobileNet V1

The evaluation metrics for the ssd_mobilenet_v1 were recorded for checkpoint 199986 and are presented in (Figure 40). In the assessment summary, the summation column represents the number of images that were evaluated for every one of the 18 classes.



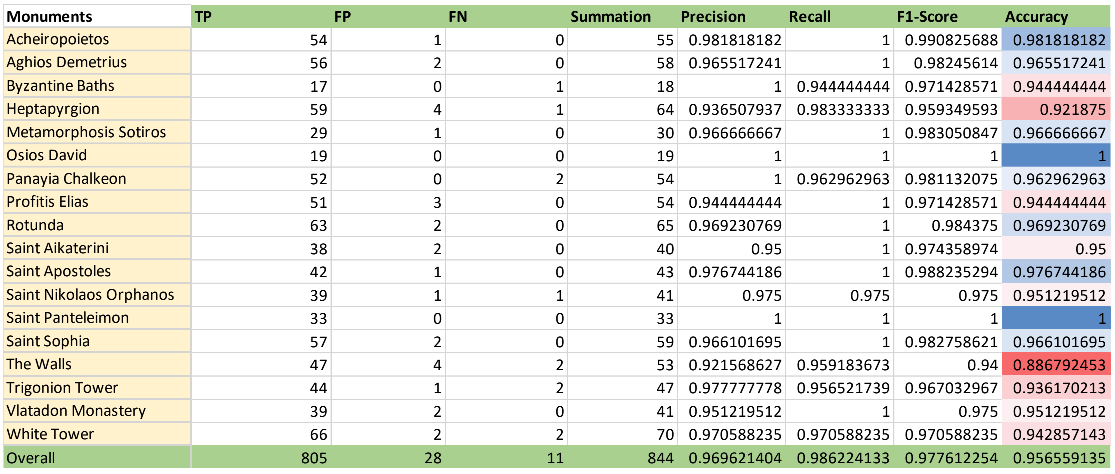
Monuments	TP	FP	FN	Summation	Precision	Recall	F1-Score	Accuracy
Acheiropoietos	53	2	0	55	0.963636364	1	0.981481481	0.963636364
Aghios Demetrius	54	4	0	58	0.931034483	1	0.964285714	0.931034483
Byzantine Baths	16	2	0	18	0.888888889	1	0.941176471	0.888888889
Heptapyrgion	53	8	3	64	0.868852459	0.946428571	0.905982906	0.828125
Metamorphosis Sotiros	29	1	0	30	0.966666667	1	0.983050847	0.966666667
Osios David	18	1	0	19	0.947368421	1	0.972972973	0.947368421
Panayia Chalkeon	52	1	1	54	0.981132075	0.981132075	0.981132075	0.962962963
Profitis Elias	51	3	0	54	0.944444444	1	0.971428571	0.944444444
Rotunda	62	2	1	65	0.96875	0.984126984	0.976377953	0.953846154
Saint Aikaterini	39	1	0	40	0.975	1	0.987341772	0.975
Saint Apostoles	40	2	1	43	0.952380952	0.975609756	0.963855422	0.930232558
Saint Nikolaos Orphanos	40	0	1	41	1	0.975609756	0.987654321	0.975609756
Saint Panteleimon	32	1	0	33	0.96969697	1	0.984615385	0.96969697
Saint Sophia	55	2	2	59	0.964912281	0.964912281	0.964912281	0.93220339
The Walls	49	2	2	53	0.960784314	0.960784314	0.960784314	0.924528302
Trigonion Tower	41	1	5	47	0.976190476	0.891304348	0.931818182	0.872340426
Vlatadon Monastery	38	3	0	41	0.926829268	1	0.962025316	0.926829268
White Tower	65	1	4	70	0.984848485	0.942028986	0.962962963	0.928571429
Overall	787	37	20	844	0.953967586	0.978996504	0.965769942	0.934554749

Figure 40: Table of the evaluation results for ssd mobilenet v1 model.

Sequentially, each metric is calculated for all the monuments, offering a more focused investigation for every respective instance. In essence, from the total amount of 844 images, the model correctly identified the 787 images, whereas 37 images were wrongly detected and in 20 the ground truth was not found, although it was present in the image. Generally, the model is able to identify the great majority of the positive cases, considering that these cases are positive, with a Recall value of almost 0.98. Conversely, the precision value varies at 0.954 mainly due to the larger number of wrong detections in comparison with the FN cases. Taking into account that the above classes do not present any significant imbalance the Accuracy values reflect these of the F1-Score, displaying a small decrease as the number of the TP plays a significant role. Overall the results are satisfactory, nevertheless, some classes require improvement, especially the “Heptapyrgion” and “Trigonion Tower” which possess a representative number of images and yet they burden the model which presents an Accuracy of 0.935.

5.2.2 SSD MobileNet V2

The same procedure was applied for the ssd_mobilenet_v2 model which was created



Monuments	TP	FP	FN	Summation	Precision	Recall	F1-Score	Accuracy
Acheiropoietos	54	1	0	55	0.981818182	1	0.990825688	0.981818182
Aghios Demetrius	56	2	0	58	0.965517241	1	0.98245614	0.965517241
Byzantine Baths	17	0	1	18	1	0.944444444	0.971428571	0.944444444
Heptapyrgion	59	4	1	64	0.936507937	0.983333333	0.959349593	0.921875
Metamorphosis Sotiros	29	1	0	30	0.966666667	1	0.983050847	0.966666667
Osios David	19	0	0	19	1	1	1	1
Panayia Chalkeon	52	0	2	54	1	0.962962963	0.981132075	0.962962963
Profitis Elias	51	3	0	54	0.944444444	1	0.971428571	0.944444444
Rotunda	63	2	0	65	0.969230769	1	0.984375	0.969230769
Saint Aikaterini	38	2	0	40	0.95	1	0.974358974	0.95
Saint Apostoles	42	1	0	43	0.976744186	1	0.988235294	0.976744186
Saint Nikolaos Orphanos	39	1	1	41	0.975	0.975	0.975	0.951219512
Saint Panteleimon	33	0	0	33	1	1	1	1
Saint Sophia	57	2	0	59	0.966101695	1	0.982758621	0.966101695
The Walls	47	4	2	53	0.921568627	0.959183673	0.94	0.886792453
Trigonion Tower	44	1	2	47	0.977777778	0.956521739	0.967032967	0.936170213
Vlatadon Monastery	39	2	0	41	0.951219512	1	0.975	0.951219512
White Tower	66	2	2	70	0.970588235	0.970588235	0.970588235	0.942857143
Overall	805	28	11	844	0.969621404	0.986224133	0.977612254	0.956559135

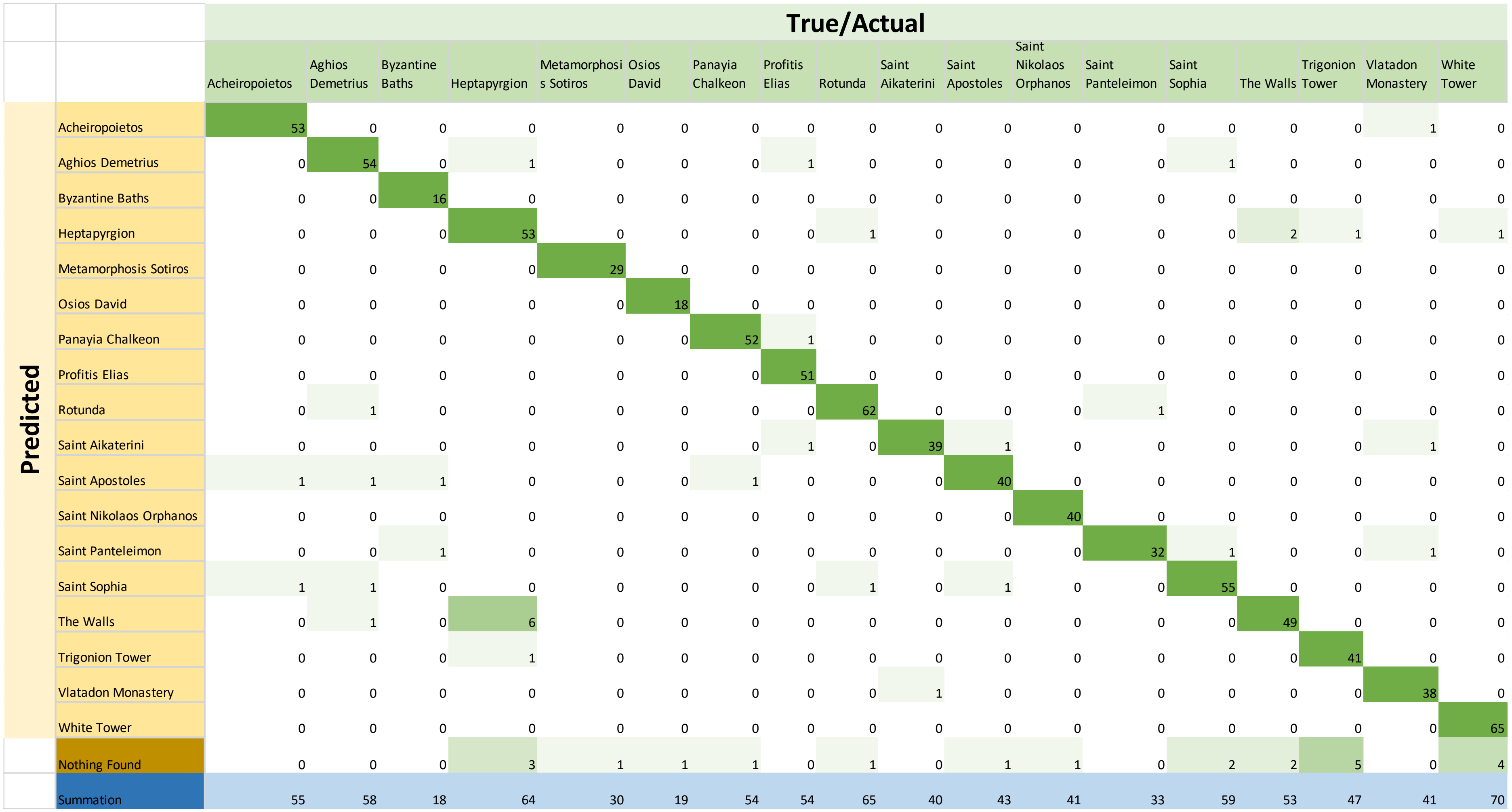
Figure 41: Table of the evaluation results for ssd mobilenet v2 model.

by checkpoint 165487, where displayed its best mAP performance. The results of the additional evaluation metrics are demonstrated in (Figure 41). At a glance, it is obvious that the ssd_mobilenet_v2 is more capable and presents improved performance regarding to the former one, in almost any single case. Out of the 844 images, it is able to recognize correctly the 805 of them, whereas 28 images were wrongly identified, and just only in 11 instances it was unable to detect the desired monument with a confidence level more than the threshold. The Precision and Recall follow the same behavior as the ssd_mobilenet_v1, with the results being superior and providing 0.97 and 0.986 accordingly. Once again the Accuracy values reflect the F1-Score ones. The model demonstrates an overall F1-Score of 0.978, something more than 1% better than the first model, and an overall Accuracy of 0.957, which is again improved by 2.2%. The lowest values in terms of F1-score and Accuracy are assigned to the “The Walls” followed by the “Heptapyrgion” and “Trigonion Tower”.

Generally, the assessment indicates an overall enhancement in the ability of recognition, for the majority of the monuments.

5.3 Confusion Matrix

In order to obtain a clearer view concerning the wrongly detected cases, in this section, we analyzed the object detection problem as a multi-classification instance, based on the concepts described in chapter 3.9.1. A confusion matrix for two investigating models is presented in (Figures 42-43). The diagonal boxes with the vivid green color represent the TP or alternatively the correctly classified cases with an IoU at least 50%. Whereas the rest of the boxes consist of the FP or the erroneous classified cases. The two tables are being completed with two additional rows, the “Nothing found” for the occasions where ground truth has not been detected (FN) and the “Summation” row which contains the total amount of the images for each monument separately. In general, the values have been scattered throughout the table with the confusion matrix of the ssd_mobilenet_v1 providing slightly higher values. However, it is significant to mention that in both cases, the two models intent to confuse the “Heptapyrgion” with “The Walls” and the opposite. In other words, it is observed a relation between the predictions of these two monuments.



		True/Actual																	
		Acheiropoietos	Aghios Demetrius	Byzantine Baths	Heptapyrgion	Metamorphosis Sotiros	Osios David	Panayia Chalkeon	Profitis Elias	Rotunda	Saint Aikaterini	Saint Apostoles	Saint Nikolaos Orphanos	Saint Panteleimon	Saint Sophia	The Walls	Trigonion Tower	Vlatadon Monastery	White Tower
Predicted	Acheiropoietos	53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	Aghios Demetrius	0	54	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0
	Byzantine Baths	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Heptapyrgion	0	0	0	53	0	0	0	0	1	0	0	0	0	0	2	1	0	1
	Metamorphosis Sotiros	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0
	Osios David	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0
	Panayia Chalkeon	0	0	0	0	0	0	52	1	0	0	0	0	0	0	0	0	0	0
	Profitis Elias	0	0	0	0	0	0	0	51	0	0	0	0	0	0	0	0	0	0
	Rotunda	0	1	0	0	0	0	0	0	62	0	0	0	1	0	0	0	0	0
	Saint Aikaterini	0	0	0	0	0	0	0	1	0	39	1	0	0	0	0	0	1	0
	Saint Apostoles	1	1	1	0	0	0	1	0	0	0	40	0	0	0	0	0	0	0
	Saint Nikolaos Orphanos	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0
	Saint Panteleimon	0	0	1	0	0	0	0	0	0	0	0	0	32	1	0	0	1	0
	Saint Sophia	1	1	0	0	0	0	0	0	1	0	1	0	0	55	0	0	0	0
	The Walls	0	1	0	6	0	0	0	0	0	0	0	0	0	0	49	0	0	0
	Trigonion Tower	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	41	0	0
	Vlatadon Monastery	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	38	0
	White Tower	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65
	Nothing Found	0	0	0	3	1	1	1	0	1	0	1	1	0	2	2	5	0	4
	Summation	55	58	18	64	30	19	54	54	65	40	43	41	33	59	53	47	41	70

Figure 42: Table of the confusion matrix for the ssd mobilenet v1 model.

		True/Actual																	
		Acheiropoietos	Aghios Demetrius	Byzantine Baths	Heptapyrgion	Metamorphosis Sotiros	Osios David	Panayia Chalkeon	Profitis Elias	Rotunda	Saint Aikaterini	Saint Apostoles	Saint Nikolaos Orphanos	Saint Panteleimon	Saint Sophia	The Walls	Trigonion Tower	Vlatadon Monastery	White Tower
Predicted	Acheiropoietos	54	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0
	Aghios Demetrius	1	56	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	Byzantine Baths	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Heptapyrgion	0	1	0	59	0	0	0	0	0	0	0	0	0	0	3	1	0	0
	Metamorphosis Sotiros	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0
	Osios David	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0
	Panayia Chalkeon	0	0	0	0	0	0	52	0	0	0	0	0	0	0	0	0	0	0
	Profitis Elias	0	1	0	0	0	0	0	51	2	2	0	0	0	0	0	0	2	0
	Rotunda	0	0	0	0	0	0	0	0	63	0	0	0	0	0	0	0	0	0
	Saint Aikaterini	0	0	0	0	1	0	0	0	0	38	0	0	0	0	0	0	0	0
	Saint Apostoles	0	0	0	1	0	0	0	1	0	0	43	0	0	1	0	0	0	0
	Saint Nikolaos Orphanos	0	0	0	0	0	0	0	0	0	0	0	39	0	0	0	0	0	0
	Saint Panteleimon	0	0	0	0	0	0	0	0	0	0	0	0	33	0	0	0	0	0
	Saint Sophia	0	0	0	0	0	0	0	0	0	0	1	0	0	57	0	0	0	0
	The Walls	0	0	0	2	0	0	0	0	0	0	0	0	0	0	47	0	0	0
	Trigonion Tower	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	44	0	1
	Vlatadon Monastery	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	40	0
	White Tower	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	66
	Nothing Found	0	0	1	1	0	0	2	0	0	0	0	1	0	0	2	2	0	2
	Summation	55	58	18	64	30	19	54	54	65	40	44	41	33	59	53	47	42	70

Figure 43: Table of the confusion matrix for the ssd mobilenet v2 model.

5.4 Comparison Examples of the SSD Mobilenet V1 and SSD V2

Indicatively, we present some images utilized to test the performance of the two models and they were not part of the training dataset. Namely, neither of the models had seen these images before. The following (Figures 44-47) provide the predicted bounding boxes and the confidence scores for each of the two models. On the left is depicted the outcome of the SSD MobileNet V1 whereas on the right the outcome of the SSD MobileNet V2 model.

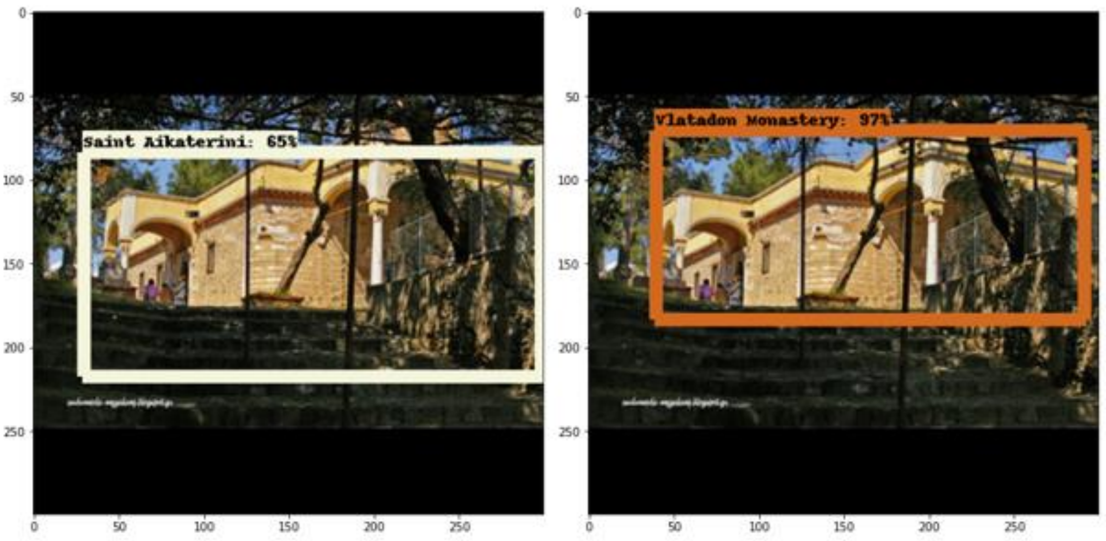


Figure 44: A nightly picture of the church of Aghios Demetrius. Both models are able to correctly identify the landmark, with the SSD MobileNet V2 being more confident for its prediction.

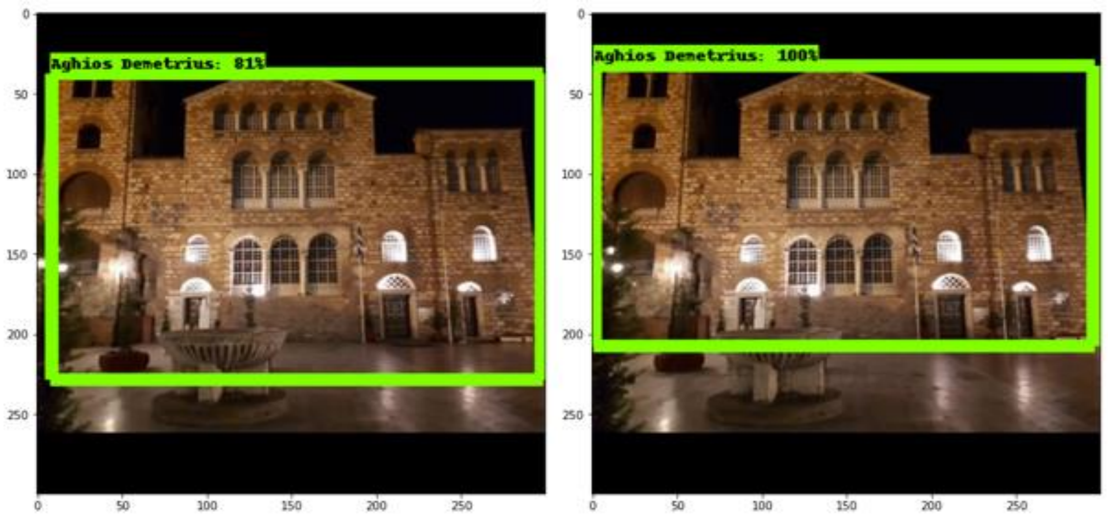


Figure 45: A picture of Vlatadon Monastery from a relatively difficult angle. Although the SSD MobileNet V2 is able to correctly pinpoint the landmark, the SSD MobileNet V1 makes a wrong detection.

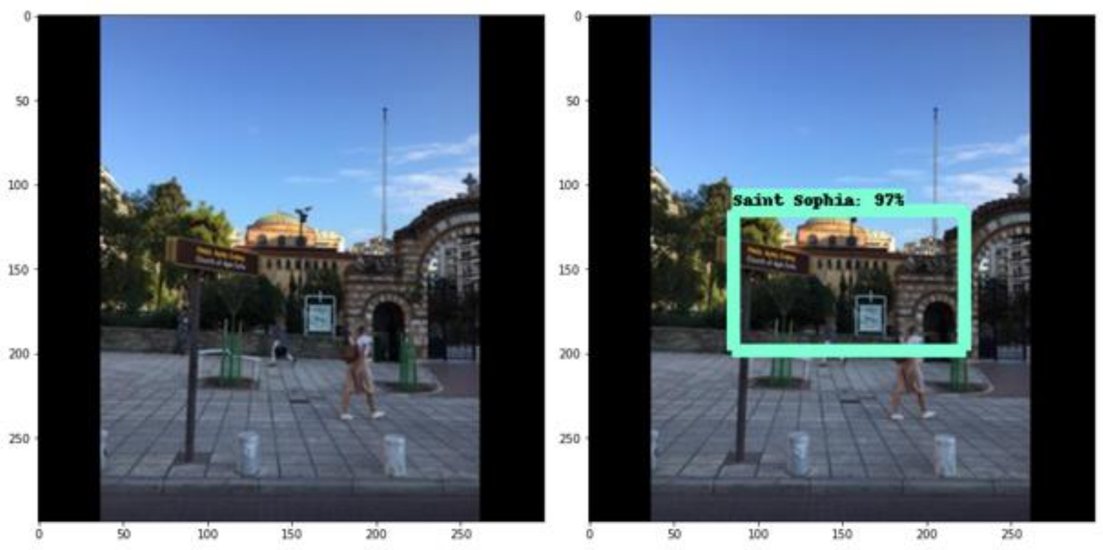


Figure 46: The above example illustrates a distant picture of the church of Saint Sophia with the sun covering the upper part of the landmark. The SSD MobileNet V1 is unable to detect the church in contrast to the SSD MobileNet V2.

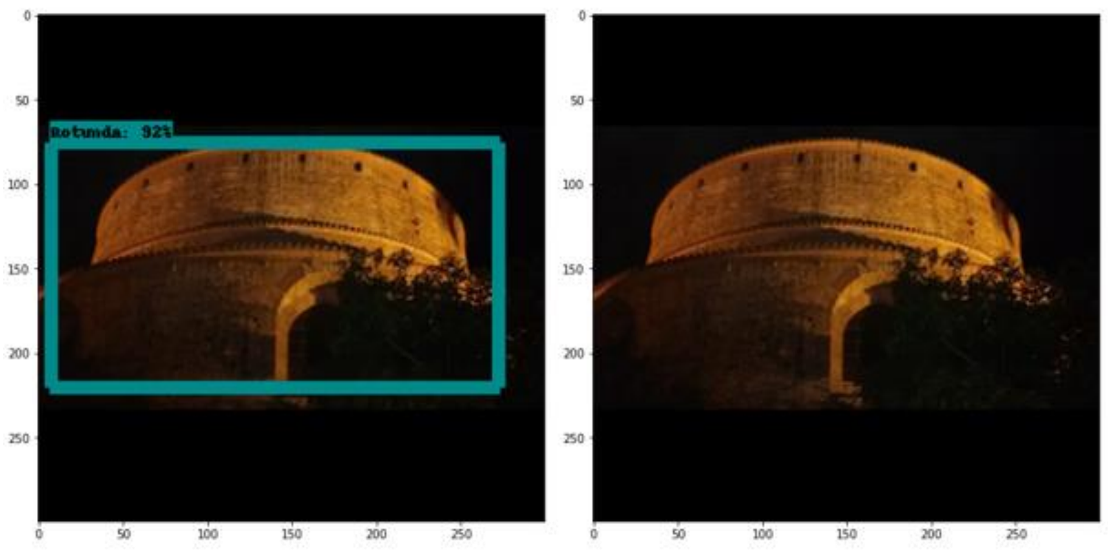


Figure 47: A nightly picture of the church of Rotunda. Despite the peculiarity of the image SSD MobileNet V1 is able to recognize the monument.

Taking into consideration the whole amount of the tested images, the SSD MobileNet V2 is characterized as a more accurate and more robust model. Although, SSD MobileNet V1 outperforms the V2 model is some specific cases, in general the latter of the two present improved capabilities. Additionally, concerning the analysis in chapter 5.1 and 5.2 the SSD MobileNet V2 not only demonstrates a higher mAP values for the three different IoU thresholds, but also (Figure 40-41) strengthen the argument that is less prone to FP and FN cases.

5.5 Applying Data Augmentation

A common method utilized especially in deep learning to enhance the results for specific tasks is that of Data Augmentation. This approach enables the production of new artificially training data based on the already existed training data. Various data augmentation techniques can be implemented for images just like position augmentation (scaling, cropping, padding, etc.) and color augmentation (brightness, saturation, etc.) offering a greater diversity to the dataset.

As object detection is the investigated subject in our case study, data augmentation should also be applied for the generated bounding boxed of the original dataset. To achieve this task, initially, all the generated (XML) files are converted into (CSV). Sequentially, the augmentation parameters are structured selecting randomly one augmenter for each instance. Furthermore, the transformation of the bounding boxes into a (data frame) format constitutes an essential part, before eventually completing the process by generating the new transformed images along with the new (CSV) file that contains the total amount of the labels (original images + augmented images).

For the purpose of this experiment scale, translate, multiplication, Gaussian Blur, Gaussian Noise, and Linear Contrast were utilized to generate the new training data. (Figure 48) demonstrates an example of the data augmentation that was applied to produce two additional images from an existing image. In addition, a total of 7550 new images were produced to supply the training process, which was conducted based on

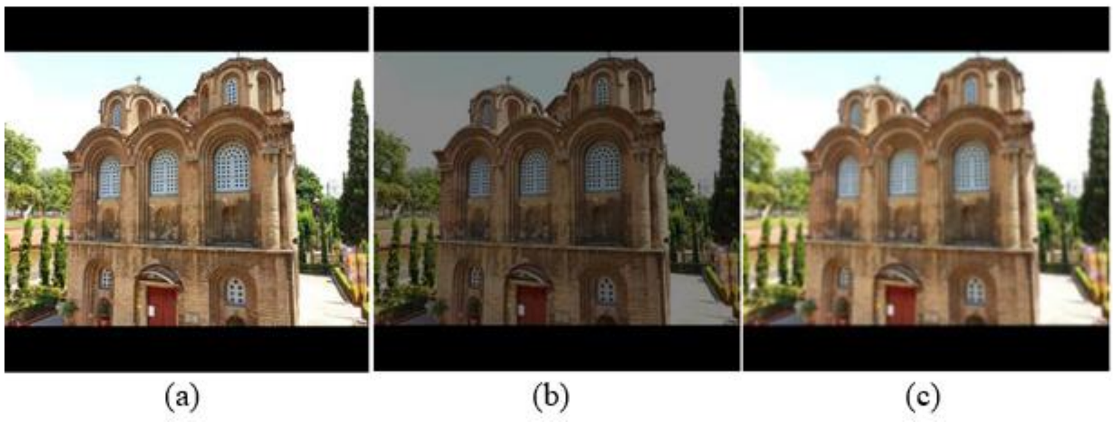
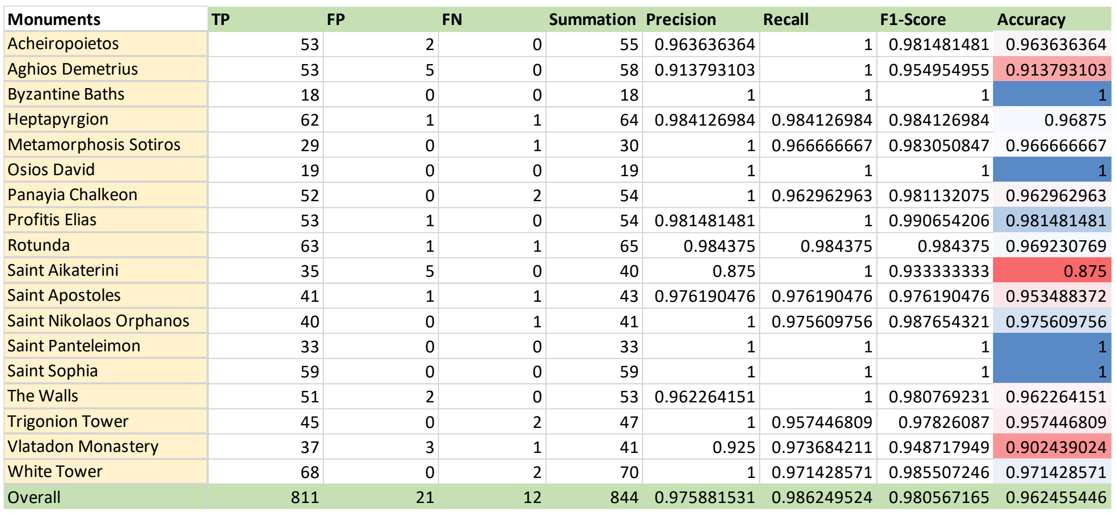


Figure 49: Example of the data augmentation technique applied. The church of Panayia Chalkeon is depicted on the three above cases. (a) It is the original image. (b) It is a newly produced image with Linear Contrast. (c) It is another newly produced image with Gaussian Blur.

Table 4: Ssd_mobilenet_V2 data augmented performance.

Step	**mAP**	**mAP_0.50_IOU**	**mAP_0.75_IOU**
1416	1.374E-01%	5.376E-01%	1.77E-03%
33378	61.49%	91.23%	69.48%
65672	65.59%	93.65%	75.10%
103870	69.37%	94.93%	79.60%
140507	71.68%	96.14%	83.31%
182708	71.14%	95.56%	82.41%
196640	73.19%	96.37%	83.37%

the SSD MobileNet V2 model, which was the one that presented overall the best outcomes. The results of the newly trained model SSD MobileNet_V2 with augmented data are shown in (Table 4). There is an apparent gradual improvement for the model as the training steps increasing exhibiting lower scores after the 196640 steps though. In this attempt, the model managed to achieve a mAP score of 73.19% which is slightly smaller than the 73.59% of the SSD MobileNet V2. Nonetheless, it is significant to mention that the mAP with IoU 0.50 presents an improvement in comparison to the Non-augmented model.



Monuments	TP	FP	FN	Summation	Precision	Recall	F1-Score	Accuracy
Acheiropoietos	53	2	0	55	0.963636364	1	0.981481481	0.963636364
Aghios Demetrius	53	5	0	58	0.913793103	1	0.954954955	0.913793103
Byzantine Baths	18	0	0	18	1	1	1	1
Heptapyrgion	62	1	1	64	0.984126984	0.984126984	0.984126984	0.96875
Metamorphosis Sotiros	29	0	1	30	1	0.966666667	0.983050847	0.966666667
Osios David	19	0	0	19	1	1	1	1
Panayia Chalkeon	52	0	2	54	1	0.962962963	0.981132075	0.962962963
Profitis Elias	53	1	0	54	0.981481481	1	0.990654206	0.981481481
Rotunda	63	1	1	65	0.984375	0.984375	0.984375	0.969230769
Saint Aikaterini	35	5	0	40	0.875	1	0.933333333	0.875
Saint Apostoles	41	1	1	43	0.976190476	0.976190476	0.976190476	0.953488372
Saint Nikolaos Orphanos	40	0	1	41	1	0.975609756	0.987654321	0.975609756
Saint Panteleimon	33	0	0	33	1	1	1	1
Saint Sophia	59	0	0	59	1	1	1	1
The Walls	51	2	0	53	0.962264151	1	0.980769231	0.962264151
Trigonion Tower	45	0	2	47	1	0.957446809	0.97826087	0.957446809
Vlatadon Monastery	37	3	1	41	0.925	0.973684211	0.948717949	0.902439024
White Tower	68	0	2	70	1	0.971428571	0.985507246	0.971428571
Overall	811	21	12	844	0.975881531	0.986249524	0.980567165	0.962455446

Figure 50: Table of the evaluation results for SSD MobileNet V2 with data augmentation technique.

To further investigate and assess the performance of the new model the same test set as that of the 5.2 was implemented. In (Figure 50) we can observe the provided results in terms of the same evaluation methods. As the table in Figure 50 suggests, interestingly the model is able to correctly identify and specify the 811 of the total amount of 844 monuments, improving the results in FP and FN. Again the model is more prone to accomplish an invalid prediction rather than making no prediction at all. By maintaining the model’s Recall value and upgrading its Precision performance the new model achieved a F1-Score of 0.98 and an Accuracy of 0.962. Despite the fact that the new models appear slightly more capable than its predecessor, it is noteworthy that it manages to increase its recognition ability on half of the monuments, sacrificing its ability for recognition in the other half monuments presenting lower scores.

6 Discussion

6.1 Quality of the data

First of all, both of the investigating models utilize a resolution of 300x300 pixels. In view of that there are models using the twice or more of this resolution, 300x300 is considered as a low one. As a result, significant information that otherwise would have gained and would have enriched the model is not obtained. Notwithstanding, often higher resolution and by extension more information from an image comes with increased computational power, something that usually nowadays mobile devices cannot offer. Consequently, as the resolution of an image is inextricably linked with the need for greater computing power, the appropriate point between these two variables should be selected in every case.

Given that the gathered images present variation regarding the lighting conditions, the ankles and the partial occlusions, generally the quality of the training data can be characterized as good. With a rough calculation, based on the fact that the dataset consists of a total of 4708 images, approximately an average number of 261 images used for each case and 80% of them or almost 209 images where used in order to train each of the 18 monuments. However, attempting to generate in a sense a 3D-model for each of these monuments with a number of 209 images, is practically impossible to cover all the possible occasions, especially for the bulkiest instances. This fact in combination with that deep learning techniques require a huge amount of information, collecting an additional considerable number of images could enhance the performance of the model.

6.2 Selecting the proper Tensorflow version

The period this thesis took place, Tensorflow had released a new object detection API based on the Tensorflow 2.3.0. Nevertheless, there are many problems attached with the procedure followed to train a model so as to be able to recognize the desired objects, mainly because of incompatibility and some bugs. For this reason, the version of Tensorflow 1.15 was utilized to perform the process, as also referred in 4.3. Both of the models were trained and tested under the older version of Tensorflow (1.15), yet providing very encouraging results. Notwithstanding, the Google’s team is working on the newer version continuously updating the already existing or creating new features and offering many new models with very promising capabilities. With several operations being enhanced and multiple models to be optimized the future belongs to the newer version of Tensorflow.

6.3 Assessment Report

As it is observed in chapter 5, both SSD MobileNet V1 and SSD MobileNet V2 start with poor scores in terms of the mAP performance since both models utilize pre-trained weights. Subsequently, in both instances and the first 20-30k steps, they have learned the vast majority of the basic parts depicted in the provided dataset. From that point and then they continue learning until they finally reach a peak, after which the scores display a downward trend.

Taking into consideration the evaluation summary from 5.2.1 and 5.2.2 there is an apparent superiority of the SSD MobileNet V2 over the V1, in almost every aspect. The second version of the SSD MobileNet is able to correctly classify and localize the desired object with an accuracy of 95.7%, 2.2% higher than its predecessor. In essence, it has the potential to identify the ground truth more efficiently and it is less vulnerable in FP and FN predictions.

Moreover, the data augmentation technique is incorporated into the experiment to further increase the number of the images that are used for training, as well as the quality of the data by adding variety to the dataset. The outcome indicated a slight overall improvement of the model trained on the new dataset, able to validly categorize the monuments with an overall accuracy of 96.25%. Notwithstanding, as aforementioned above, the interesting part is that although the newly trained model presents marginally better results and improves the scores for half of the monuments the SSD MobileNet V2 that trained on the original dataset is more suitable for recognizing the other half monuments. The reason behind this occurrence may be induced by various factors. Of course, the quality of the original data plays a fundamental role, not only for the results of the data augmentation but also for the results of the two selected models. Apart from this fact, there are numerous methods to augment an image, each one separately, or in combination, providing absolutely different outcomes for the same image. In practice, different kinds of objects in regard to the shape, the color, the possible form, and many other elements, require a different approach.

Concerning the operation of the application on Android, it is fully functional and the outcome is in harmony with the results provided in chapter 5. The model is able to identify the majority of the objects even for difficult angles with reasonable effectiveness in a short period, offering a smooth functioning. However, it is a common phenomenon, the appearance of FP predictions, or alternatively wrong identifications of objects that do not present any similarity with the predicted one. Possibly, the problem stems from the transfer learning concept itself. Although the model performs satisfactorily recognizing and localizing the objects, it is not trained in many real-world instances with various backgrounds and get confused in several previously unseen cases. In other words, this type of model that is used with pre-trained weights rationally should also be trained in objects that are irrelevant with the desired ones, or differently in negative images.

7 Future Work

7.1 Using GPS to aid Recognition

Tracking and correctly classifying an object is often a challenging and a difficult task, especially considering the numerous factors that are involved in a such type of process. Even well trained models, in some cases fail to provide adequately accurate, valid and precise data. A technology that all the mobile devices incorporate and could substantially reduce the failures on recognition, is that of the Global Positioning System commonly known as GPS. Receiving information through the 4 existing GPS satellites, this feature enables us to obtain valuable data about the geolocation and the time of a device, operating independently and without influencing any other operation. The concept behind this idea is to combine the models capabilities with the reliability of GPS. With this way, by providing access to GPS, the application will immediately restrict significantly the area, excluding those monuments that are located far from the given location and reducing dramatically the chance of a wrong detection. Typically, with the recent developments particularly in the technological field, the detection of a position using this feature can be achieved even with accuracy within a few meters. Therefore, fusing GPS with an object detection model may lead to a near infallible model.

7.2 Providing Information about the Monument/Landmark

A simple notion to comprehend yet useful, especially for people interested at learning about diverse cultures, broaden their horizons and acquire a multifaceted personality, is the provision of additional information, with a median they can promptly interact, such as the personal mobile device. The concept behind this notion is to implement a button in the application’s interface that will be accessible during a correct recognition. With this way, a user will have the potential for supplementary information concerning the monument located right in front of him, just like the physical attributes, historical data, gallery with images or even some external links.

8 Conclusions

In this thesis, we have provided an approach to constructing and implementing a Monument/Landmark object detection model based on Convolution Neural Networks. Gathering a sufficient number of images for the UNESCO World Heritage Monuments of Thessaloniki and using the SSD MobileNet V1 and V2 models, we were able to successfully train and apply them in an Android application, capable of conducting a real-time object detection. Utilizing the Transfer Learning technique, both models are able to identify and localize the desired objects with satisfactory accuracy, achieving at the same time high mAP performance. The application operates properly performing predictions consisted of a bounding box that encloses the detected object in combination with a score that describes the confidence of the model. Considering that the models provide accurate results in a short time of period, it strengthens the fact that real-time object detection can be also applied on mobile devices, even with limited computational power. The results presented in chapter 5 confirm the high performance for the investigating models, proving synchronously a superiority of the SSD MobileNet V2 over the SSD MobileNet V1. Moreover, as reasonably proven different monument exhibit different results, with some of them being easier and some other more challenging to identify. The data augmentation technique implemented on the V2 model managed to present a slight increase in the overall accuracy of the model.

To further enrich the effort presented in this thesis, collecting additional images under various circumstances and from diverse angles of view, particularly for those with lower metrics, would definitely benefit the performance of the models. Combining this action with the incorporation of the GPS feature into the application, it would reduce the erroneous detections to the minimum and upgrade the overall outcome. Of course, offering the potential for learning crucial information about each monument with a build-in function is an action that can only be beneficial.

Although, a few years back object detection models were either accuracy-oriented or speed-oriented, nowadays there is the potential to achieve both, without sacrificing any of these factors and actually without using any substantial computing power. Computer vision and by extent object detection is something that has received great attention and has widely studied among many researchers. With the rapidly increasing development of technology and the constant improvements of the object detection models, the future of the object recognition is bright.

Bibliography

[1] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z., 2016. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).

[2] Szeliski, R., 2010. Computer vision: algorithms and applications. Springer Science & Business Media.

[3] Umbaugh, S.E., 2010. Digital image processing and analysis: human and computer vision applications with CVIPtools. CRC press.

[4] Alsing, O., 2018. Mobile object detection using tensorflow lite and transfer learning.

[5] Gada, S., Mehta, V., Kanchan, K., Jain, C. and Raut, P., 2017, December. Monument recognition using deep neural networks. In 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC) (pp. 1-6). IEEE.

[6] Palma, Valerio. (2019). TOWARDS DEEP LEARNING FOR ARCHITECTURE: A MONUMENT RECOGNITION MOBILE APP. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLII-2/W9. 551-556. 10.5194/isprs-archives-XLII-2-W9-551-2019.

[7] Termritthikun, C., Kanprachar, S. and Muneesawang, P., 2018. NU-LiteNet: Mobile Landmark Recognition using Convolutional Neural Networks. arXiv preprint arXiv:1810.01074.

[8] Shi, H., Xu, M. and Li, R., 2017. Deep learning for household load forecasting—A novel pooling deep RNN. IEEE Transactions on Smart Grid, 9(5), pp.5271-5280.

[9] Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J. and Keutzer, K., 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. arXiv preprint arXiv:1602.07360.

[10] Saini, A., Gupta, T., Kumar, R., Gupta, A.K., Panwar, M. and Mittal, A., 2017, December. Image based Indian monument recognition using convoluted neural networks. In 2017 International Conference on Big Data, IoT and Data Science (BID) (pp. 138-142). IEEE.

[11] Harel, J., Koch, C. and Perona, P., 2007. Graph-based visual saliency. In Advances in neural information processing systems (pp. 545-552).

[12] Triantafyllidis, G. and Kalliatakis, G., 2013. Image based monument recognition using graph based visual saliency. ELCVIA: electronic letters on computer vision and image analysis, 12(2), pp.88-97.

[13] Ninawe, A., Mallick, A.K., Yadav, V., Ahmad, H., Sah, D.K. and Barna, C., 2018, September. Cathedral and Indian Mughal Monument Recognition Using Tensorflow. In International Workshop Soft Computing Applications (pp. 186-196). Springer, Cham.

[14] Gupta, U. and Chaudhury, S., 2015, December. Deep transfer learning with ontology for image classification. In 2015 Fifth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG) (pp. 1-4). IEEE.

[15] Huang, T., 1996. Computer vision: Evolution and promise.

[16] Rosenfeld, A., 1983. Quadtrees and Pyramids: hierarchical representation of images. In Pictorial data analysis (pp. 29-42). Springer, Berlin, Heidelberg.

[17] Barron, J.T. and Malik, J., 2014. Shape, illumination, and reflectance from shading. IEEE transactions on pattern analysis and machine intelligence, 37(8), pp.1670-1687.

[18] Kass, M., Witkin, A. and Terzopoulos, D., 1988. Snakes: Active contour models. International journal of computer vision, 1(4), pp.321-331.

[19] Geman, S. and Geman, D., 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Transactions on pattern analysis and machine intelligence, (6), pp.721-741.

[20] Moons, T., Van Gool, L. and Vergauwen, M., 2009. 3D Reconstruction from Multiple Images: Principles. Now Publishers Inc.

[21] Feng, X., Jiang, Y., Yang, X., Du, M. and Li, X., 2019. Computer vision algorithms and hardware implementations: A survey. Integration, 69, pp.309-320.

[22] Fagnant, D.J. and Kockelman, K., 2015. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. Transportation Research Part A: Policy and Practice, 77, pp.167-181.

[23] Ingenta Connect. 2016. Google Translate App. [https://tinyurl.com/ycse9mq9] Accessed 15 October 2020.

[24] Sipe, M.A., Schneiderman, H.W. and Nechyba, M.C., Google LLC, 2013. Facial recognition. U.S. Patent 8,457,367.

[25] Milletari, F., Navab, N. and Ahmadi, S.A., 2016, October. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In 2016 fourth international conference on 3D vision (3DV) (pp. 565-571). IEEE.

[26] Alpaydin, E., 2020. Introduction to machine learning. MIT press.

[27] Smola, A. and Vishwanathan, S.V.N., 2008. Introduction to machine learning. Cambridge University, UK, 32(34), p.2008.

[28] Stefano Zanero and Sergio M. Savaresi. 2004. Unsupervised learning techniques for an intrusion detection system. In Proceedings of the 2004 ACM symposium on Applied computing (SAC '04). Association for Computing Machinery, New York, NY, USA, 412–419. DOI:https://doi.org/10.1145/967900.967988

[29] Z. A. Zhao and H. Liu. Spectral Feature Selection for Data Mining. Chapman & Hall/CRC Press, Virginia Beach, VA, 2012.

[30] Ongsulee, P., 2017, November. Artificial intelligence, machine learning and deep learning. In 2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE) (pp. 1-6). IEEE.

[31] Sharifi, Y., Moghbeli, A., Hosseinpour, M. and Sharifi, H., 2019. Study of Neural Network Models for the Ultimate Capacities of Cellular Steel Beams. Iranian Journal of Science and Technology, Transactions of Civil Engineering, pp.1-11.

[32] Yegnanarayana, B., 2009. Artificial neural networks. PHI Learning Pvt. Ltd.

[33] Graupe, D., 2013. Principles of artificial neural networks (Vol. 7). World Scientific.

[34] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2017. Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6), pp.84-90.

[35] Rawat, W. and Wang, Z., 2017. Deep convolutional neural networks for image classification: A comprehensive review. Neural computation, 29(9), pp.2352-2449.

[36] Mechelli, A. and Vieira, S. eds., 2019. Machine Learning: Methods and Applications to Brain Disorders. Academic Press.

[37] Millstein, F., 2020. Convolutional Neural Networks In Python: Beginner's Guide To Convolutional Neural Networks In Python. Frank Millstein.

[38] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence, 37(9), pp.1904-1916.

[39] Gong, Y., Wang, L., Guo, R. and Lazebnik, S., 2014, September. Multi-scale orderless pooling of deep convolutional activation features. In European conference on computer vision (pp. 392-407). Springer, Cham.

[40] Rawat, W. and Wang, Z., 2017. Deep convolutional neural networks for image classification: A comprehensive review. Neural computation, 29(9), pp.2352-2449.

[41] Yuan, B., 2016, September. Efficient hardware architecture of softmax layer in deep neural network. In 2016 29th IEEE International System-on-Chip Conference (SOCC) (pp. 323-326). IEEE.

[42] Khan, A., Sohail, A., Zahoora, U. and Qureshi, A.S., 2019. A survey of the recent architectures of deep convolutional neural networks. Artificial Intelligence Review, pp.1-62.

[43] Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y., 2016. Deep learning (Vol. 1, No. 2). Cambridge: MIT press.

[44] Zhao, Z.Q., Zheng, P., Xu, S.T. and Wu, X., 2019. Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems, 30(11), pp.3212-3232.

[45] Zou, Z., Shi, Z., Guo, Y. and Ye, J., 2019. Object detection in 20 years: A survey. arXiv preprint arXiv:1905.05055.

[46] K. E. Van de Sande, J. R. Uijlings, T. Gevers, and A. W. Smeulders, "Segmentation as selective search for object recognition," in Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011, pp. 1879–1886.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual 29 recognition," in European conference on computer vision. Springer, 2014, pp. 346–361.

[48] R. Girshick, "Fast r-cnn," in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

[49] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in Advances in neural information processing systems, 2015, pp. 91–99.

[50] T.-Y. Lin, P. Dollar, R. B. Girshick, K. He, B. Hariharan, ´ and S. J. Belongie, "Feature pyramid network

[51] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.

[52] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in European conference on computer vision. Springer, 2016, pp. 21–37.

[53] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, ´ "Focal loss for dense object detection," IEEE transactions on pattern analysis and machine intelligence, 2018.

[54] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y. and Berg, A.C., 2016, October. Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

[55] Viola, P. and Jones, M., 2001. Robust real-time object detection. International journal of computer vision, 4(34-47), p.4.

[56] Jiao, L., Zhang, F., Liu, F., Yang, S., Li, L., Feng, Z. and Qu, R., 2019. A survey of deep learning-based object detection. IEEE Access, 7, pp.128837-128868.

[57] Tensorflow. [https://www.tensorflow.org/]. Accessed 24 October 2020.

[58] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. and Kudlur, M., 2016. Tensorflow: A system for large-scale machine learning. In 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16) (pp. 265-283).

[59] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M. and Ghemawat, S., 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

[60] Tensorflow. Deploy machine learning models on mobile and IoT devices. [https://www.tensorflow.org/lite/]. Accessed 24 October 2020.

[61] Louis, M.S., Azad, Z., Delshadtehrani, L., Gupta, S., Warden, P., Reddi, V.J. and Joshi, A., 2019. Towards deep learning using tensorFlow lite on RISC-V. In Third Workshop on Computer Architecture Research with RISC-V (CARRV).

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

[63] Weiss, K., Khoshgoftaar, T.M. and Wang, D., 2016. A survey of transfer learning. Journal of Big data, 3(1), p.9.

[64] Pan, S.J. and Yang, Q., 2009. A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), pp.1345-1359.

[65] Mark Everingham et. al. The PASCAL Visual Object Classes Homepage. [http://host.robots.ox.ac.uk/pascal/VOC/]. Accessed 30 October 2020.

[66] CadaLab. COCO Detection Challenge. [https://competitions.codalab.org/competitions/5181]. Accessed 30 October 2020.

[67] Googleleapis. Overview of the Open Images Challenge 2018. [https://storage.googleapis.com/openimages/web/challenge.html]. Accessed 30 October 2020.

[68] Sokolova, M., Japkowicz, N. and Szpakowicz, S., 2006, December. Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation. In Australasian joint conference on artificial intelligence (pp. 1015-1021). Springer, Berlin, Heidelberg.

[69] Veropoulos, K., Campbell, C. and Cristianini, N., 1999, July. Controlling the sensitivity of support vector machines. In Proceedings of the international joint conference on AI (Vol. 55, p. 60).

[70] Kaggle. Imagenet object detection challenge. [https://www.kaggle.com/c/imagenet-object-detection-challenge]. Accessed 30 October 2020.

[71] Padilla, R., Netto, S.L. and da Silva, E.A., 2020, July. A survey on performance metrics for object-detection algorithms. In 2020 International Conference on Systems, Signals and Image Processing (IWSSIP) (pp. 237-242). IEEE.

[72] Everingham, M., Van Gool, L., Williams, C.K., Winn, J. and Zisserman, A., 2010. The pascal visual object classes (voc) challenge. International journal of computer vision, 88(2), pp.303-338.

[73] COCO. [https://cocodataset.org/#detection-eval]. Accessed 30 October 2020.

[74] Lipton, Z.C., Elkan, C. and Narayanaswamy, B., 2014. Thresholding classifiers to maximize f1 score. arXiv preprint ArXiv:1402.1892, 14.

[75] Python Software Foundation. [https://pypi.org/project/labelImg/1.4.0/]. Accessed 25 November 2020.

[76] Google Colaboraotry. Welcome to Colaboratory. [https://colab.research.google.com/notebooks/intro.ipynb]. Accessed 25 November 2020.

[77] Bisong, E., 2019. Google Colaboratory. In Building Machine Learning and Deep Learning Models on Google Cloud Platform (pp. 59-64). Apress, Berkeley, CA.

[78] Google Drive. [https://www.google.com/drive/]. Accessed 25 November 2020.

[79] GitHub. [https://github.com/tensorflow/models/tree/master/research]. Accessed 5 December 2020.

[80] GitHub. [https://tinyurl.com/yd75w6j9]. Accessed 5 December 2020.

[81] GitHub. TensorFlow 1 Detection Model Zoo. [https://tinyurl.com/yd75w6j9]. Accessed 5 December 2020.

[82] COCO. [https://cocodataset.org/#home]. Accessed 7 December 2020.

[83] GitHub. Supported object detection evaluation protocols. [https://tinyurl.com/yd6x479h]. Accessed 7 December 2020.

[84] Tensorflow. TensorBoard: TensorFlow's visualization toolkit. [https://tinyurl.com/y8wumqek]. Accessed 7 December 2020.

[85] Tensorflow. Get started with TensorFlow Lite. [https://www.tensorflow.org/lite/guide/get_started]. Accessed 7 December 2020.

[86] Google Developers. Android Studio. [https://developer.android.com/studio?authuser=1]. Accessed 7 December 2020.

[87] Wolfson, M. and Felker, D., 2013. Android developer tools essentials: Android Studio to Zipalign. " O'Reilly Media, Inc.".

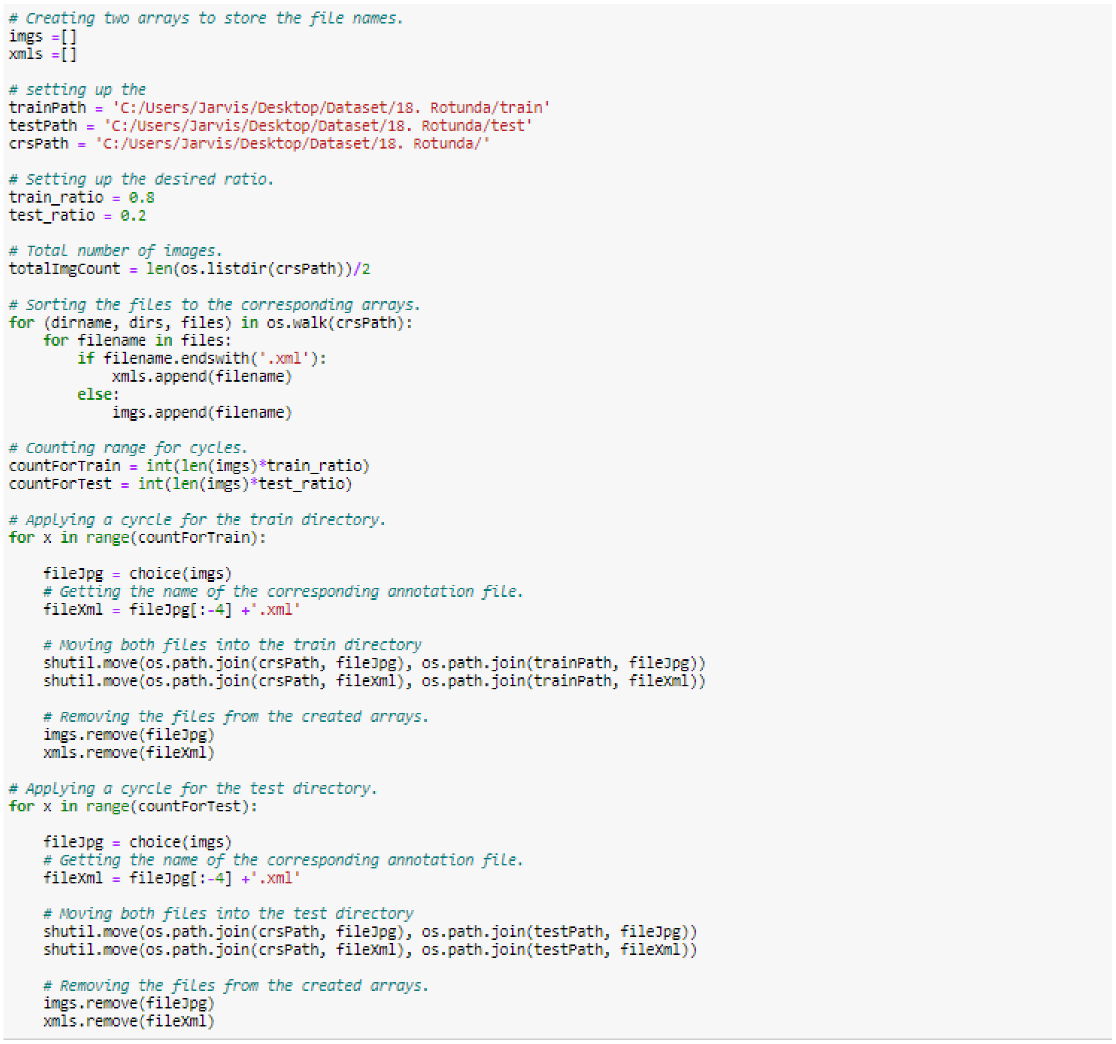
[88] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Appendix

In this section, the parts of the effort that are of technical interest are presented. It also supplements the contribution and the conducted experiment in chapter 4.

1. Splitting the dataset into training and testing set

A significant step before the training phase is to divide the dataset into a training set that will be used to train a model and a testing set that is going to be utilized to evaluate the performance of the model. In view of the fact, that is this stage the images have been annotated it is also essential to separate the generated (XML) files accordingly, as



```
# Creating two arrays to store the file names.
imgs =[]
xmls =[]

# setting up the
trainPath = 'C:/Users/Jarvis/Desktop/Dataset/18. Rotunda/train'
testPath = 'C:/Users/Jarvis/Desktop/Dataset/18. Rotunda/test'
crsPath = 'C:/Users/Jarvis/Desktop/Dataset/18. Rotunda/'

# Setting up the desired ratio.
train_ratio = 0.8
test_ratio = 0.2

# Total number of images.
totalImgCount = len(os.listdir(crsPath))/2

# Sorting the files to the corresponding arrays.
for (dirname, dirs, files) in os.walk(crsPath):
    for filename in files:
        if filename.endswith('.xml'):
            xmls.append(filename)
        else:
            imgs.append(filename)

# Counting range for cycles.
countForTrain = int(len(imgs)*train_ratio)
countForTest = int(len(imgs)*test_ratio)

# Applying a cyrcle for the train directory.
for x in range(countForTrain):

    fileJpg = choice(imgs)
    # Getting the name of the corresponding annotation file.
    fileXml = fileJpg[:-4] +'.xml'

    # Moving both files into the train directory
    shutil.move(os.path.join(crsPath, fileJpg), os.path.join(trainPath, fileJpg))
    shutil.move(os.path.join(crsPath, fileXml), os.path.join(trainPath, fileXml))

    # Removing the files from the created arrays.
    imgs.remove(fileJpg)
    xmls.remove(fileXml)

# Applying a cyrcle for the test directory.
for x in range(countForTest):

    fileJpg = choice(imgs)
    # Getting the name of the corresponding annotation file.
    fileXml = fileJpg[:-4] +'.xml'

    # Moving both files into the test directory
    shutil.move(os.path.join(crsPath, fileJpg), os.path.join(testPath, fileJpg))
    shutil.move(os.path.join(crsPath, fileXml), os.path.join(testPath, fileXml))

    # Removing the files from the created arrays.
    imgs.remove(fileJpg)
    xmls.remove(fileXml)
```

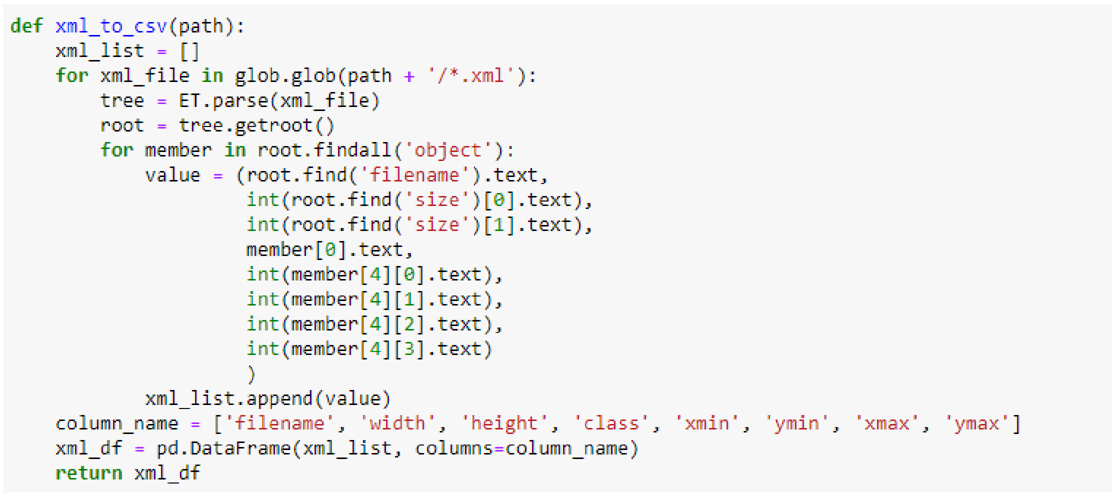
Figure 51: Example of the block of code used to separate the images and the corresponding XML files into train and test set.

shown in, Figure 51. Selecting randomly the train and test images with a ratio of 0.8

and 0.2 accordingly, the process continues with the implementation of the same procedure for the annotated files.

2. Applying data augmentation

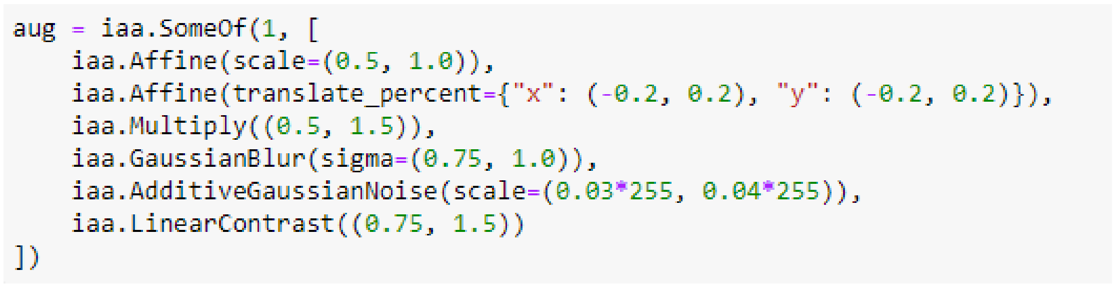
As aforementioned, in order to apply data augmentation, it is first vital to create a (CSV) file for the original annotated dataset. Of course, this process should only be implemented on the training set, as it is the set we desire to generate more images from. By providing the appropriate path as it is demonstrated in Figure 52, the following function "xml_to_csv" pinpoints all the (XML) files for a given path directory, reads the recorded attributes, and generates a new (CSV) file. Sequentially, we setup the



```python
def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df
```

Figure 51: Function that converts the XML file in a direction into a CSV file.

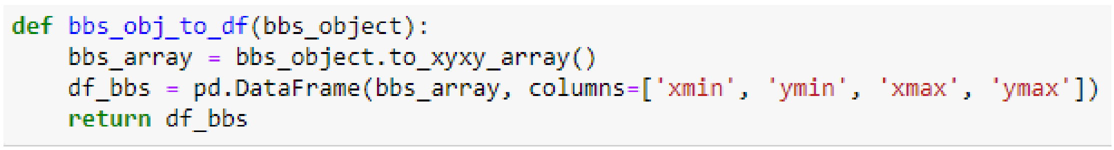
augmentation parameters selecting one random parameter for each case, as displayed in Figure 52. An important step before implementing data augmentation is to create a



```python
aug = iaa.SomeOf(1, [
    iaa.Affine(scale=(0.5, 1.0)),
    iaa.Affine(translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)}),
    iaa.Multiply((0.5, 1.5)),
    iaa.GaussianBlur(sigma=(0.75, 1.0)),
    iaa.AdditiveGaussianNoise(scale=(0.03*255, 0.04*255)),
    iaa.LinearContrast((0.75, 1.5))
])
```

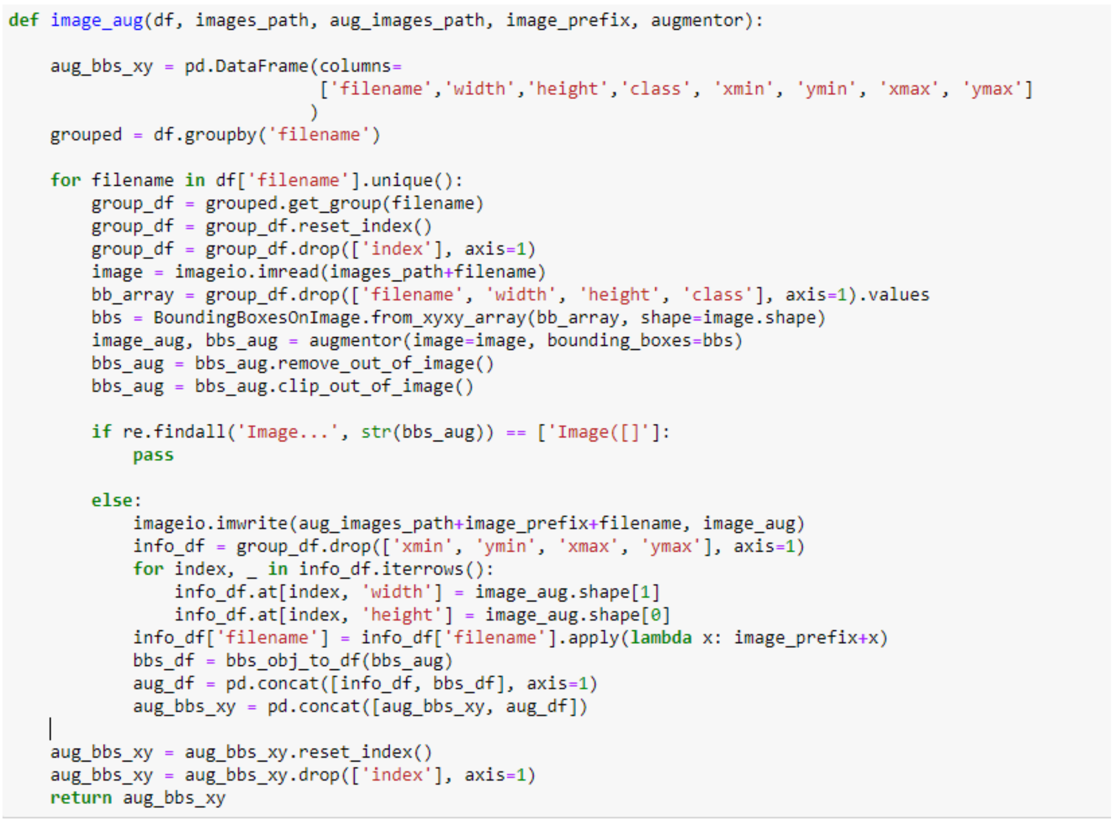
Figure 52: Setting up the parameters that will be used for the augmentation.

function that is able to convert the bounding boxes into an array format and then into a data frame format which is necessary for the easier manipulation of the wanted attributes, Figure 53.



```python
def bbs_obj_to_df(bbs_object):
    bbs_array = bbs_object.to_xyxy_array()
    df_bbs = pd.DataFrame(bbs_array, columns=['xmin', 'ymin', 'xmax', 'ymax'])
    return df_bbs
```

Figure 53: Converting the bounding boxes into a data frame.



```python
def image_aug(df, images_path, aug_images_path, image_prefix, augmentor):

    aug_bbs_xy = pd.DataFrame(columns=
                              ['filename','width','height','class', 'xmin', 'ymin', 'xmax', 'ymax']
                             )
    grouped = df.groupby('filename')

    for filename in df['filename'].unique():
        group_df = grouped.get_group(filename)
        group_df = group_df.reset_index()
        group_df = group_df.drop(['index'], axis=1)
        image = imageio.imread(images_path+filename)
        bb_array = group_df.drop(['filename', 'width', 'height', 'class'], axis=1).values
        bbs = BoundingBoxesOnImage.from_xyxy_array(bb_array, shape=image.shape)
        image_aug, bbs_aug = augmentor(image=image, bounding_boxes=bbs)
        bbs_aug = bbs_aug.remove_out_of_image()
        bbs_aug = bbs_aug.clip_out_of_image()

        if re.findall('Image...', str(bbs_aug)) == ['Image([]']:
            pass

        else:
            imageio.imwrite(aug_images_path+image_prefix+filename, image_aug)
            info_df = group_df.drop(['xmin', 'ymin', 'xmax', 'ymax'], axis=1)
            for index, _ in info_df.iterrows():
                info_df.at[index, 'width'] = image_aug.shape[1]
                info_df.at[index, 'height'] = image_aug.shape[0]
            info_df['filename'] = info_df['filename'].apply(lambda x: image_prefix+x)
            bbs_df = bbs_obj_to_df(bbs_aug)
            aug_df = pd.concat([info_df, bbs_df], axis=1)
            aug_bbs_xy = pd.concat([aug_bbs_xy, aug_df])

    aug_bbs_xy = aug_bbs_xy.reset_index()
    aug_bbs_xy = aug_bbs_xy.drop(['index'], axis=1)
    return aug_bbs_xy
```

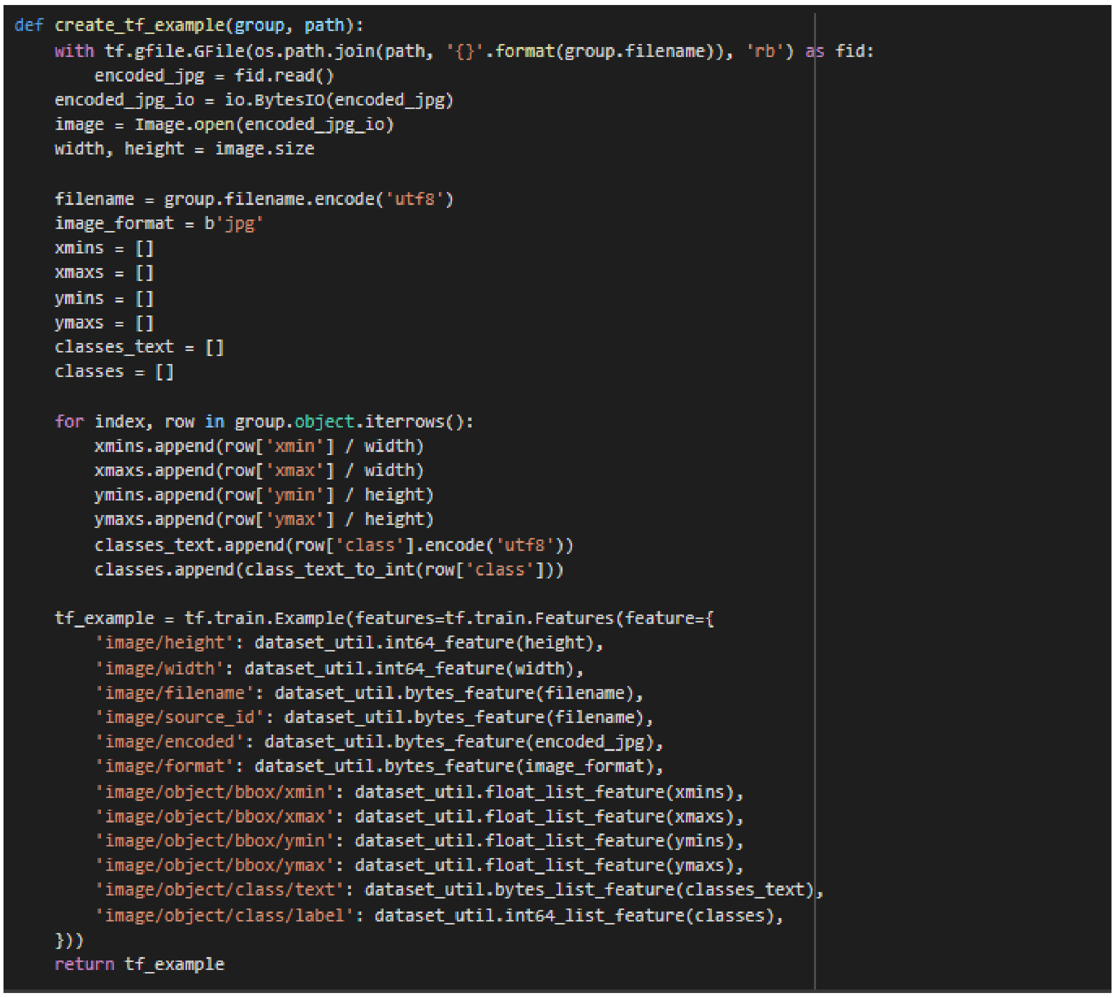
Figure 54: Function that carries out data augmentation for both the images and the corresponding bounding boxes.

With the function “image_aug” and by providing the originally created data frame, the path directory of the images, a new path directory to store the augmented images, a prefix to separate the images, as well as an augmenter, we are able to generate a new set of augmented images based on the original train set. Again, the “bbs_obj_to_df” function contributes to the proper update of the new bounding boxes that depends on the respective new image.

In the final step, a single “concat” command is used to combine the original (CSV) with the new one that contains the new labels.

3. Converting xml files to tfrecord

As aforementioned in chapter 4.4, in order to convert an (XML) file into a (record) format, it is vital to first transform it into a (CSV). This statement could be accomplished by following the same procedure as that of the previous stage, with “xml_to_csv” function. Subsequently, with the “create_tf_example” function by inserting the group and the desired path and by selecting a ‘utf8’ encode format and ‘jpg’ image format, 6 empty lists are created to store the (record) format attributes of each case, as it is illustrated in Figure 55.



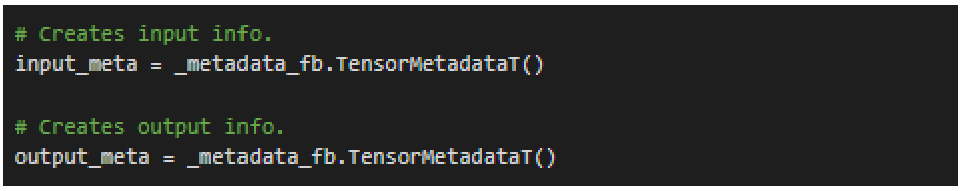
```python
def create_tf_example(group, path):
    with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }))
    return tf_example
```

Figure 55: Function that converts a CSV into a tf_example.

With the correct completion of this process, literally, an image takes the form of a huge set of characters that are meaningless for the human.

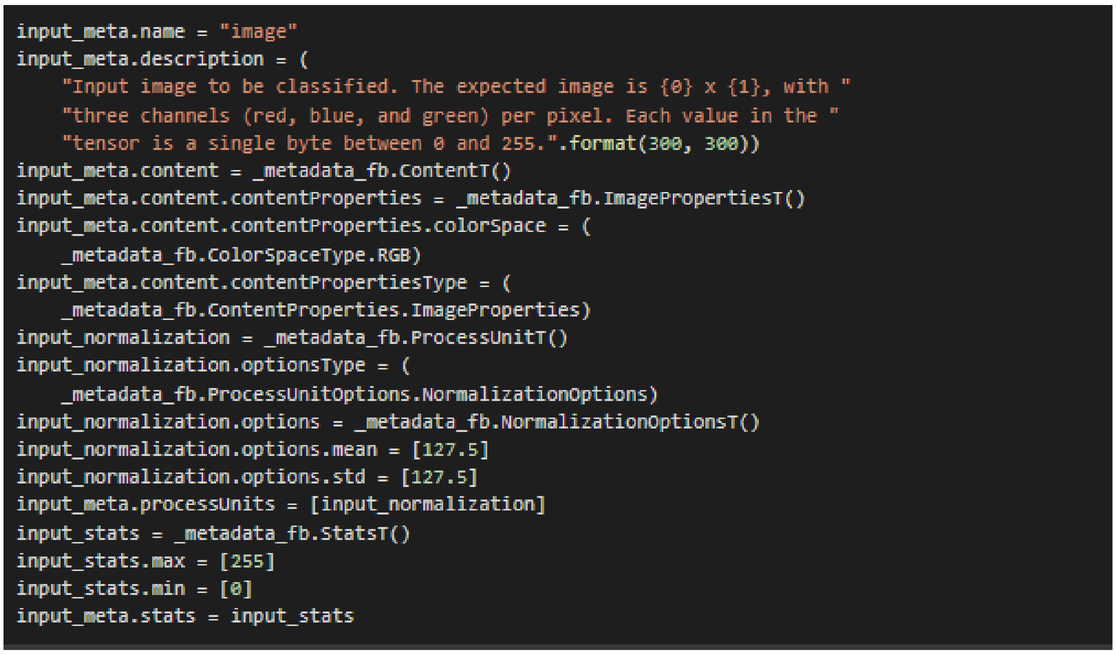
4. Adding a metadata file to the transformed model

As also referred to in 4.11 a final stage before the deployment of the model into Android is the creation of a metadata file. This file contains both human and computer readable parts and it is essential since specifies vital attributes like the resolution of the images and the utilized classes. To execute this process, the library “tflite-support” should be installed. Figures 56- demonstrate the blocks of code that were implemented to generate a metadata file for a specified model.



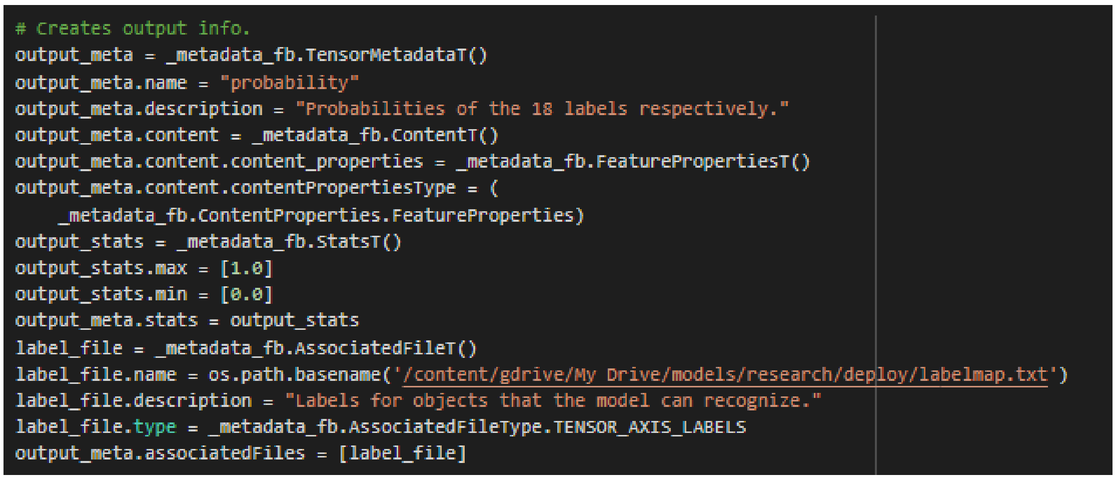
```python
# Creates input info.
input_meta = _metadata_fb.TensorMetadataT()

# Creates output info.
output_meta = _metadata_fb.TensorMetadataT()
```

Figure 56: Creating input and output information.



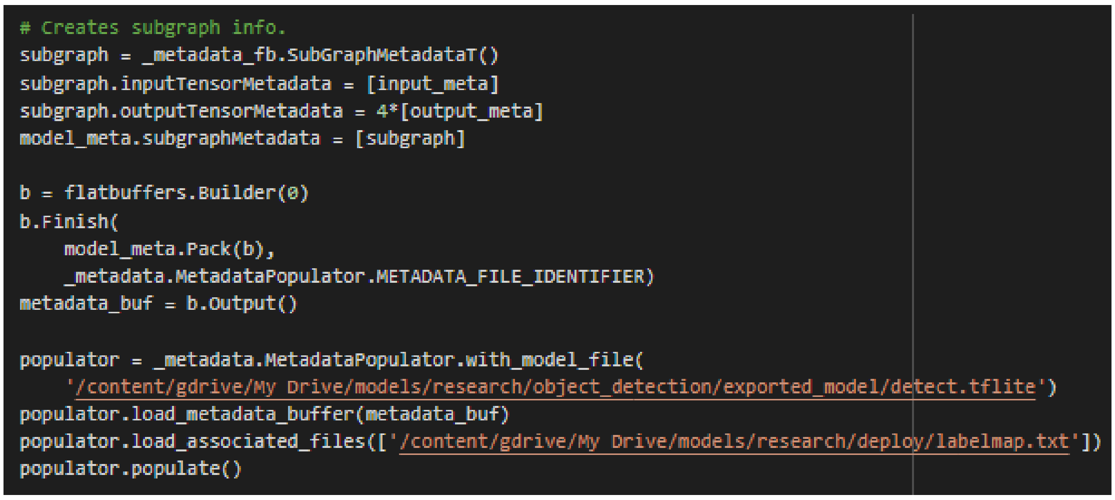
```python
input_meta.name = "image"
input_meta.description = (
    "Input image to be classified. The expected image is {0} x {1}, with "
    "three channels (red, blue, and green) per pixel. Each value in the "
    "tensor is a single byte between 0 and 255.".format(300, 300))
input_meta.content = _metadata_fb.ContentT()
input_meta.content.contentProperties = _metadata_fb.ImagePropertiesT()
input_meta.content.contentProperties.colorSpace = (
    _metadata_fb.ColorSpaceType.RGB)
input_meta.content.contentPropertiesType = (
    _metadata_fb.ContentProperties.ImageProperties)
input_normalization = _metadata_fb.ProcessUnitT()
input_normalization.optionsType = (
    _metadata_fb.ProcessUnitOptions.NormalizationOptions)
input_normalization.options = _metadata_fb.NormalizationOptionsT()
input_normalization.options.mean = [127.5]
input_normalization.options.std = [127.5]
input_meta.processUnits = [input_normalization]
input_stats = _metadata_fb.StatsT()
input_stats.max = [255]
input_stats.min = [0]
input_meta.stats = input_stats
```

Figure 57: Using an image as an input type.



```python
# Creates output info.
output_meta = _metadata_fb.TensorMetadataT()
output_meta.name = "probability"
output_meta.description = "Probabilities of the 18 labels respectively."
output_meta.content = _metadata_fb.ContentT()
output_meta.content.content_properties = _metadata_fb.FeaturePropertiesT()
output_meta.content.contentPropertiesType = (
    _metadata_fb.ContentProperties.FeatureProperties)
output_stats = _metadata_fb.StatsT()
output_stats.max = [1.0]
output_stats.min = [0.0]
output_meta.stats = output_stats
label_file = _metadata_fb.AssociatedFileT()
label_file.name = os.path.basename('/content/gdrive/My Drive/models/research/deploy/labelmap.txt')
label_file.description = "Labels for objects that the model can recognize."
label_file.type = _metadata_fb.AssociatedFileType.TENSOR_AXIS_LABELS
output_meta.associatedFiles = [label_file]
```

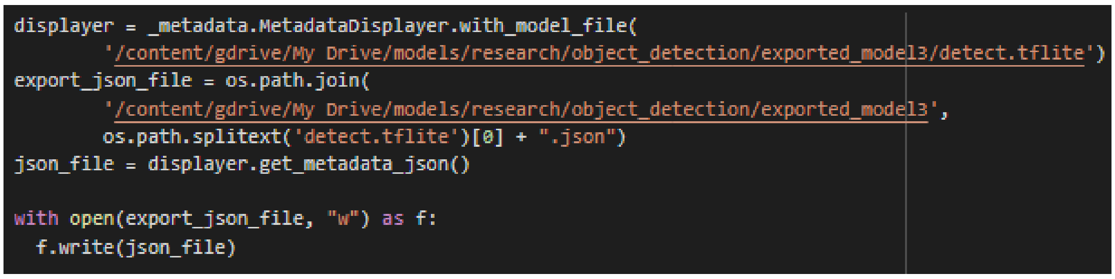
Figure 58: Mapping labels to an output tensor.



```python
# Creates subgraph info.
subgraph = _metadata_fb.SubGraphMetadataT()
subgraph.inputTensorMetadata = [input_meta]
subgraph.outputTensorMetadata = 4*[output_meta]
model_meta.subgraphMetadata = [subgraph]

b = flatbuffers.Builder(0)
b.Finish(
    model_meta.Pack(b),
    _metadata.MetadataPopulator.METADATA_FILE_IDENTIFIER)
metadata_buf = b.Output()

populator = _metadata.MetadataPopulator.with_model_file(
    '/content/gdrive/My Drive/models/research/object_detection/exported_model/detect.tflite')
populator.load_metadata_buffer(metadata_buf)
populator.load_associated_files(['/content/gdrive/My Drive/models/research/deploy/labelmap.txt'])
populator.populate()
```

Figure 59: Combining the input and the output information and using “populate” method to write the files into a TFLite file.

Optionally, “MetadataDisplayer” could be exploited in order to visualize the created metadata file for the desired model or even to generate a (JSON) file.



```python
displayer = _metadata.MetadataDisplayer.with_model_file(
        '/content/gdrive/My Drive/models/research/object_detection/exported_model3/detect.tflite')
export_json_file = os.path.join(
        '/content/gdrive/My Drive/models/research/object_detection/exported_model3',
        os.path.splitext('detect.tflite')[0] + ".json")
json_file = displayer.get_metadata_json()

with open(export_json_file, "w") as f:
  f.write(json_file)
```

Figure 60: Visualizing the data.

5. Hardware Specification

For the experiment, the model was trained on Google Colab with the following hardware specifications.

- GPU: Tesla T4
- CPU: Intel(R) Xeon(R) CPU @ 2.20GHz (1 core, 2 threads)
- RAM: 13GB of maximum available ram
- Hard Disk: 37GB of maximum available storage

Every user has the potential to run its project on this computational server for 12 hours a day.