ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
Τμήμα Ηλεκτρολόγων & Ηλεκτρονικών Μηχανικών

UNIVERSITY of WEST ATTICA
FACULTY OF ENGINEERING
Department of Electrical & Electronics Engineering

Πρόγραμμα Μεταπτυχιακών Σπουδών
Διαδικτυωμένα Ηλεκτρονικά Συστήματα

Master of Science in
Internetworked Electronic Systems

MSc Thesis

Implementing an automated graffiti detection system utilizing deep learning algorithms

Student: Tzitamidis, Alexander, Registration Number: IES-0047
MSc Thesis Supervisor: Charalampos, Patrikakis, Associate Professor

ATHENS-EGALEO, SEPTEMBER 2019

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
Τμήμα Ηλεκτρολόγων & Ηλεκτρονικών Μηχανικών

UNIVERSITY of WEST ATTICA
FACULTY OF ENGINEERING
Department of Electrical & Electronics Engineering

Πρόγραμμα Μεταπτυχιακών Σπουδών
Διαδικτυωμένα Ηλεκτρονικά Συστήματα

Master of Science in
Internetworked Electronic Systems

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη συστήματος αυτόματης αναγνώρισης graffiti χρησιμοποιώντας αλγόριθμους βαθιάς μάθησης

Μεταπτυχιακός Φοιτητής: Τζιταμίδης, Αλέξανδρος, ΑΜ: IES-0047
Επιβλέπων: Χαράλαμπος Πατρικάκης, Αναπληρωτής Καθηγητής

ΑΙΓΑΛΕΩ, ΣΕΠΤΕΜΒΡΙΟΣ 2019

ABSTRACT

In the scope of this thesis, Machine Learning technologies are studied upon, as well together with hardware and software frameworks that support each technology. The practical goal is to make an automated graffiti tag detection application, with which, the user can use a smartphone to detect graffiti tags in real-time through the use of the device's camera. The development of such application involves effort in creating a large dataset containing images with graffiti tags taken in different environments, as well as annotating each image with bounding boxes indicating the location and size of each tag. The dataset is adapted to the requirements of the software framework used. Various machine learning techniques are used, such as transfer learning to minimize training time and improve accuracy, of the detection system in order to attempt to get the best experience possible from it.

KEYWORDS: Graffiti Tags, Deep Learning, Machine Learning, Object Detection, Transfer Learning

ΠΕΡΙΛΗΨΗ

Στο πλαίσιο αυτής της εργασίας, διερευνώνται και μελετώνται οι τεχνολογίες Μηχανικής Μάθησης, καθώς και τα πακέτα λογικσμικού και υλικού που υποστηρίζουν την κάθε τεχνολογία. Στόχος ειναι να φτιαχτεί μια εφαρμογή αυτόματης ανίχνευσης ταγκιών (graffiti), με την οποία ο χρήστης μπορεί με ένα κινητό τηλέφωνο να ανισχεύση ταγκιές σε πραγματικό χρόνο μέσω της χρήσης της κάμερας της συσκευής. Η ανάπτυξη μιας τέτοιας εφαρμογής εμπλέκει την δημιουργία ενός μεγάλου συνόλου δεδομένων, το οποίο εμπεριέχει εικόνες με ταγκιές τραβηγμένες σε μια πληθώρα συνηθηκών, όπως επίσης την καταγραφή σε κάθε εικόνα των κουτιών οριοθέτησης ως ένδειξη της θέσης και του μεγέθους της κάθε ταγκιάς. Το σύνολο δεδομένων προσαρμόζεται ανάλογα με τις απαιτήσεις των χρησιμοποιούμενου πακέτου λογισμικού. Διάφορες τεχνικές μηχανικής μάθησης χρησιμοποιούνται, όπως μεταφορά μάθησης για την ελαχιστοποίηση του χρόνου εκμάθησης και την βελτιστοποίηση της αποτελεσματικότητας, του συστήματος ανίχνευσης με επιδίωξη της καλύτερης δυνατής εμπειρίας που μπορεί να παρέχει αυτό.

ΛΕΞΕΙΣ – ΚΛΕΙΔΙΑ: Ανίχνευση Αντικειμένων, Βαθιά Μάθηση, Μηχανική Μάθηση, Ταγκιές, Μεταφορά Μάθησης

TABLE OF CONTENTS

Introduction

Prologue

The environment around us is constantly evolving with machine learning algorithms becoming intelligent every day. Machines learn from the information that was collected by humans in the past, as well as from information that are generated in the present. In this era, a large portion of the human population is seeking for a way to make use of this technology. Although the concept of machine learning was introduced in the 1900s, it didn't flourish as the gathered data wasn't enough to reach meaningful conclusions. Also, the processing power of the hardware wasn't enough to achieve satisfactory results in a respectable amount of time. With today's hardware and data, machine learning can be used to automate simple and complex daily tasks. This technology is being already utilized in many areas, such as marketing, where machine learning algorithms recommend specific products to audience with an increased chance of purchasing them. Digital assistants, such as Google Assistant, Alexa and Siri, are ever-growing virtual assistants utilizing machine learning to answer to so many possible questions a user might have. Autonomous driving is also a very big field, making cars require the assistance of a human driver less necessary over time. Machine learning algorithms can be found in mail servers. Whenever a user opens his/her e-mail, there is a good chance that irrelevant mails (usually used for advertising or scamming purposes) can be found in the inbox that the user has not signed-up for. In the past, many users often would spent daily 10-20 minutes every day just to clean their inbox from such mails. With the advances of machine learning, that problem does not exist anymore as algorithms auto-flag which mails are classified as "spam", which automatically get sent to a "spam" folder, leaving the inbox always clutter-free. Considering how many people make use of an e-mail every day, not performing this tedious task is a massive saving of time and effort. It is clear that machine learning has its applications in many areas, making the everyday life easier by solving a variety of problems. Another issue is graffiti vandalism, where people paint spray to draw artwork, marks, writing on public streets to display their skills or get a message across. These can be indications of increased crime rate (such as theft of paint sprays) because of a certain rebellious community in an area.

Purpose

Graffiti is an issue that has existed for decades. The reason graffiti has been an unpleasant problem is because it is an act of marking with writing, symbols and drawings other people's property without their consent, which is illegal in most countries. However, little to no actions have been made to the people that commit them as it doesn't cause direct / immediate concerns to the community, partially because graffiti is often thought of as art. Machine learning is one way of many to combat this long-standing problem. The purpose of the thesis is to make a system that is able to perceive as accurately as possible of what a graffiti tag is and thus, able to automatically detect one when it sees it. Using machine learning, this is an attempt to reduce the amount of new graffiti tags popping up every day on someone else’s property. As a graffiti detection system cannot directly prevent the vandals from committing their crime, it can be used to reverse the damage caused in an effective manner.

Structure

In the first chapter of this thesis, artificial intelligence and its underlying technologies are presented. Firstly, the definition of artificial intelligence and a short history how it started and progressed till the present time. The chapter will focus more into its machine learning subset explaining many different approaches that one can take to utilize this technology. Some approaches are more beneficial for specific applications, while others to other applications one can have in mind. The chapter digs deeper into approaches and techniques, such as deep learning and transfer learning, that will prove beneficial in developing the specific object detection system of this thesis, which is a graffiti tag detection application. In the end of the chapter, some of the many frameworks (each having its own strong points) are presented that enable the usage of machine learning in a fast and efficient manner.

In the second chapter, the technical section of this project is unveiled. It starts with the explanation of the aim of the project, which is followed by its requirements in software and hardware. It continues with a guide fully explaining the methology of each step in order, from the very first, where the collection of the data takes place, till the very last, where the executable file of the application is created. Each step contains the necessary scripts (with explanations) and graphs to fully grasp the workflow.

In the third chapter, the prototype application is presented. A guide how to launch it for the first time is shown and how easy it is to use. The workflow of the application itself is presented via a block diagram. Also, through the use of the application some sample images are shown to estimate the performance of the graffiti detection system with a first glance.

Finally, in the last chapter some conclusions regarding the result of the project are expressed. The conclusions focus on the evaluation of the detection system, its limitations and some thoughts / suggestions on what should have been different in the development process to achieve better results. Along with these conclusions, use-cases of the graffiti detection system are recommended, along with possible future improvements to extend the functionality of the application.

Artificial Intelligence

Definition

Artificial Intelligence (AI) is the capability of a machine to mimic human behavior. Specifically, to mimic human behavior an “intelligent” computer must be able to learn by acquiring information and using it according to a set of rules, reason by reaching conclusions and self-correct. Main components of human intelligence are considered to be the process of learning, reasoning, problem solving, perception and using language [1].

- *Learning* in artificial intelligence can be approached in various ways. The simplest way is by trial and error. A computer playing a puzzle can iterate through a number of actions towards a solution and choose the one that gets it closest to the solution. This action can be memorized so it can be re-used when the computer finds itself in the same circumstances. An important aspect of learning is the ability to generalize the problem. For example, the computer might learn that the plural of the word “cat” corresponds to “cats”. By generalizing the problem, it must be able to understand that by adding an “s” at the end of the word it is able to achieve the plural of (most) nouns.
- *Reasoning* is the ability to reach conclusions according to the data available to you. Conclusions can be categorized as deductive or inductive. Deductive reason is commonly used in sciences like mathematics, while inductive reasoning is used in sciences, where data is collected and models are created to fit that data.
- *Problem solving* in artificial intelligence is often the process of iterating through a number of possible actions in order to reach the pre-defined solution. It is a tentative process as it requires large amount of data and time when the problem is complex.
- *Perception* is a means of gathering information. Nowadays, there are many advanced sensors that can collect data from the environment with sufficient accuracy and speed. Computers use that data to fragment the environment information into different objects with various relationships between each other. An important aspect of this process is to understand if an object is the same as another if perceived with different conditions (e.g. lighting, angle of view).
- *Language* is a system of signs that have meaning. In order for an “intelligent” computer to communicate it needs to correspond its information to a set of signs, which the other party can understand the same way as the computer does. It is important to note that signs have meaning by convention and thus, can differ (e.g. from region to region, language to language).

Artificial Intelligence is incorporated into a variety of types of technology. Machine Learning is a major part of the AI field, having Deep Learning as one of its subsets and being the technology appropriate for the application of this study.

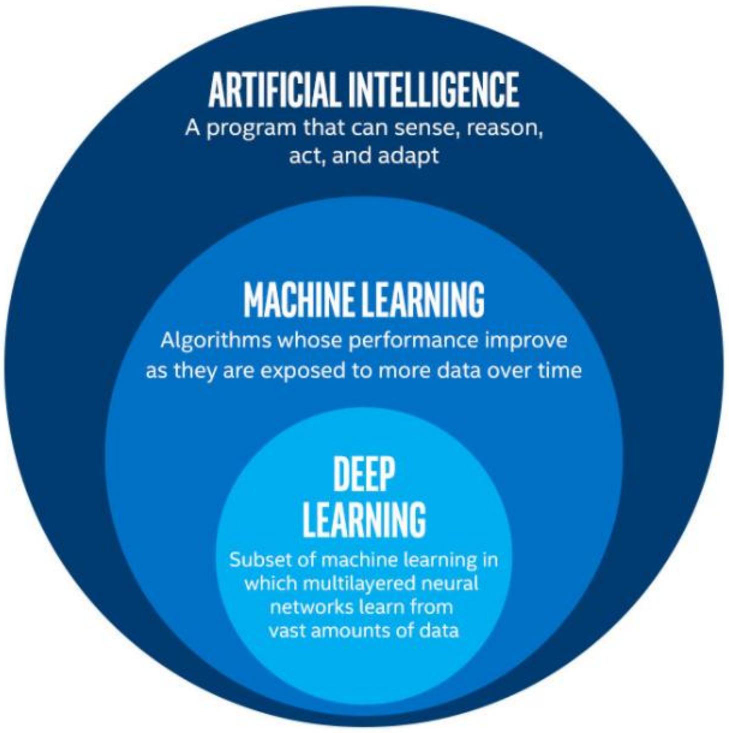

ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

Figure 1: Artificial Intelligence, Machine Learning and Deep Learning [2]

History

The first occurrence of a neural network was when mathematician Walter Pitts and neurologist Warren McCulloch wrote a paper of how human neurons might work. They created a working model using electrical circuits [3, 4]. In 1950, Alan Turing created the well-known Turing Test, which a machine can pass if it manages to "fool", through conversation, a person into thinking that the computer is human and not a machine. In 1952, Arthur Samuel created a program, which was helping an IBM computer improve at playing checkers on runtime. Board games are often used in machine learning as they are understandable by humans and can become very complex at the same time [5, 6, 7, 8]. A few years later, Frank Rosenblatt designed an artificial neural network. It was called "Perceptron" and its goal was pattern/shape recognition. In the next year, 1959, Bernard Widrow and Marcian Hoff created a model called "ADELINE" in Stanford University. It was able to detect binary patterns in a stream of bits in order to predict what the next bit would be. The next generation of this model was called "MADELINE", which could detect and eliminate echoes over phone lines. This application is still in use today [3, 9]. Until late 1970s, there wasn't any significant progress due to the popularity of the Von Neumann architecture, where logic and data of the programs were stored in the same memory, which was simpler to understand than the complexity of the neural networks. In 1979, a graduate student of Stanford University created the “Stanford Cart”, which was able to navigate a room, full of chairs, without human assistance [10, 11].

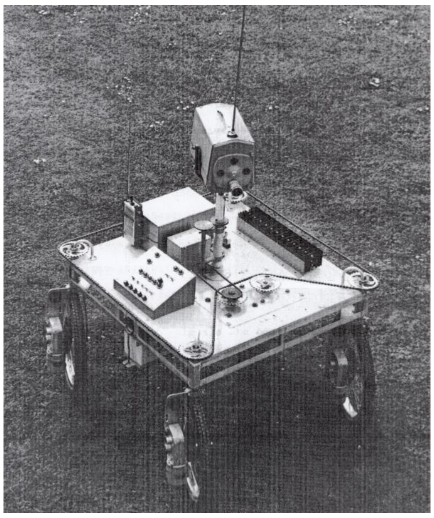


Figure 2: Stanford Cart with radio links [12]

In the 1980s, John Hopfield suggested the creation of a network with bidirectional lines, Japan focused on advanced neural networks and NETtalk was invented by Terry Sejnowski and Charles Rosenberg. NETtalk was capable of teaching itself how to pronounce new words. In one week, it was able to learn 20,000 new words [13, 14]. Three researchers from the Stanford psychology department extended the work of Widrow and Hoff to develop neural networks with multiple layers, allowing them to learn over a long period of time. In 1997, IBM’s computer “Deep Blue” came into existence. This computer was a chess-playing computer, which managed to win against the chess champion, Garry Kasparov [15, 16].

Figure 3: IBM’s Deep Blue [17]

In the next 2 years, AT&T Bell laboratories researched on digit recognition for detecting hand-written postcodes from the Postal Service with a sufficient accuracy and the University of Chicago developed the CAD Prototype Intelligent Workstation to diagnose cancer 52% more accurately than radiologists did at that time [18]. In the 21st century, machine learning starts being used in commercial applications. In 2006, Netflix announced a competition, which offered $1M for whoever beats its algorithm at predicting consumer film ratings. This competition incentivized a lot of researchers in focusing in the machine learning field. Three years later, the BellKor's Pragmatic Chaos team of AT&T won the prize, beating the second team by a few minutes [19, 20]. In 2011, another IBM computer, Watson, beats two Jeopardy! champions [21].

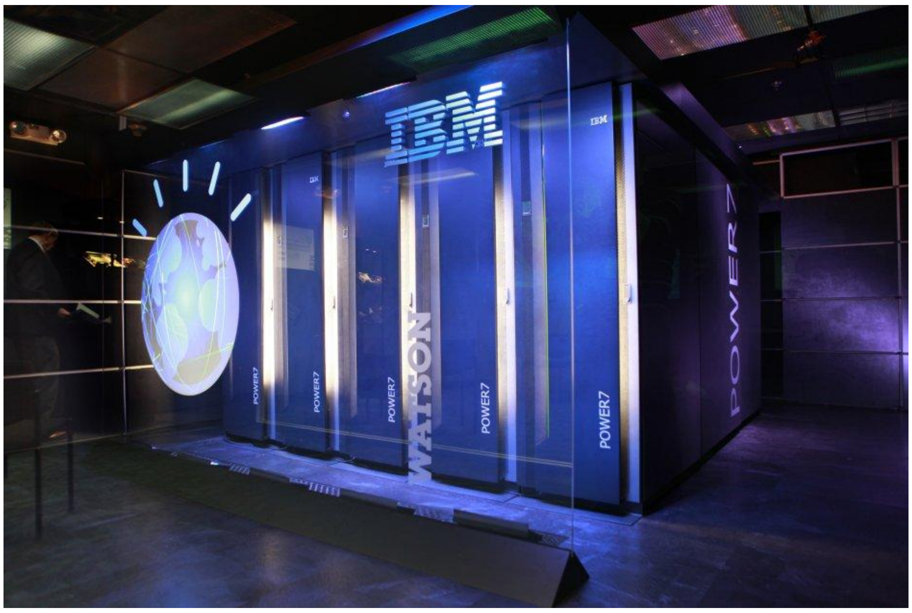


Figure 4: IBM's Watson [22]

Google Brain, a deep neural network, was created by Jeff Dean of Google. It focused on pattern detection in images and videos, more specifically it learned to recognize humans and felines with 81.7% and 74.8% accuracy respectively [23, 24]. Because the neural network had access to Google's massive amount of data, it was superior to other neural networks at the time. In 2014, Chatbot "Eugene Goostman" passed the Turing Test by convincing 33% of the human judges that it was a Ukrainian teen [25] and computers improve the ER (Emergency Room) experience in hospitals by using simulation to predict ER wait times based on data like staff size, medical history and hospital structural layout [26, 27]. In 2015, Google's "AlphaGo" beats a professional player at "Go", which is considered to be the most difficult board game worldwide [28, 29]. In the same year, detectives defined common characteristics of criminal behavior when it comes to fraud and money laundering in order for a machine learning program to learn and detect fraudulent users on the PayPal site [30, 31]. In 2016, an Oxford team trained an artificial-intelligence system to identify lip-read words with an accuracy of 93.4% [32]. The clothing manufacturer, North Face, partners up with IBM to integrate Watson's natural language processing in a mobile app. The app acts as a human sales associate to help consumer find what products they're looking for [33, 34].

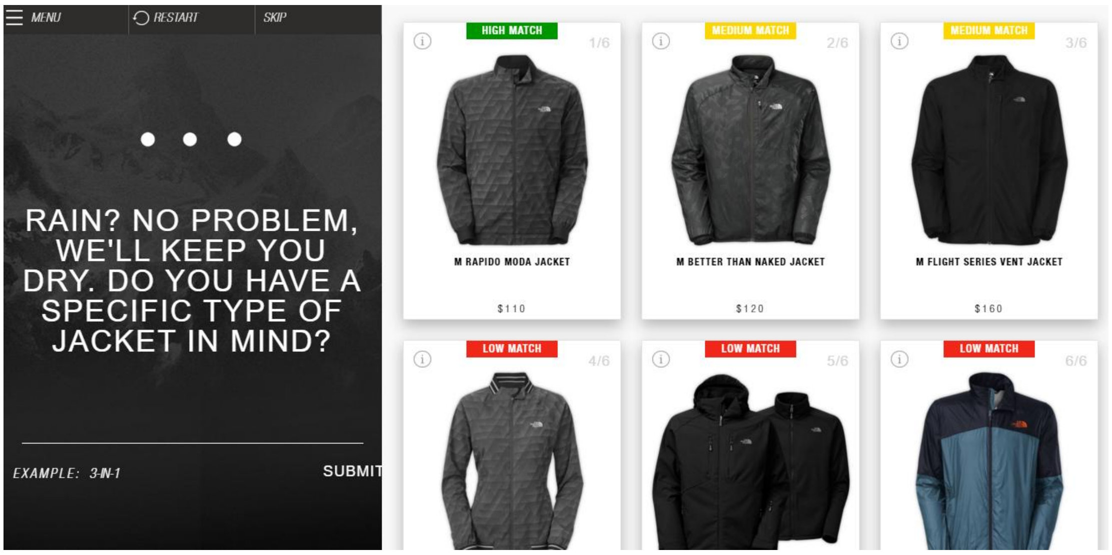


Figure 5: North Face's "sale associate" using IBM's Watson [35]

In 2017, the Jigsaw team from Alphabet trains a system by providing millions of website comments as input in an attempt to prevent online harassment when moderation resources were limited [36, 37]. Machine learning algorithms have made their way into countless other experiences, such as IoT to boost security and gathering biometric data to predict/diagnose disease [38].

Machine Learning

The ability of systems to automatically learn performing a particular task and improve from experience without being explicitly programmed is called machine learning. This technology entails algorithms that use "training data" in order to look for patterns in the provided data. This training process is an attempt to generalize the problem and make better predictions in the future with new data.

There are many types of machine learning algorithms, each having a distinct approach. They vary on the type of data they receive as input and the type of data they output. Each type algorithm might be more suitable at certain problem-solving tasks. The most popular types of machine learning approaches are:

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

Supervised Learning

In supervised learning, the algorithm is fed with labeled data, which means that the algorithms know both the input and the corresponding desired output it should reach. In the learning process, the algorithm uses the input without taking into account its labeled output in the beginning and produces a result as its prediction, which is finally compared with the right output (from the labeling process). The deviation between the predicted and the right output is used to adjust the model accordingly in an attempt to correct it and then, the learning process repeats in the next iteration. The model uses the learned patterns to estimate the output of new data, whose label is unknown.

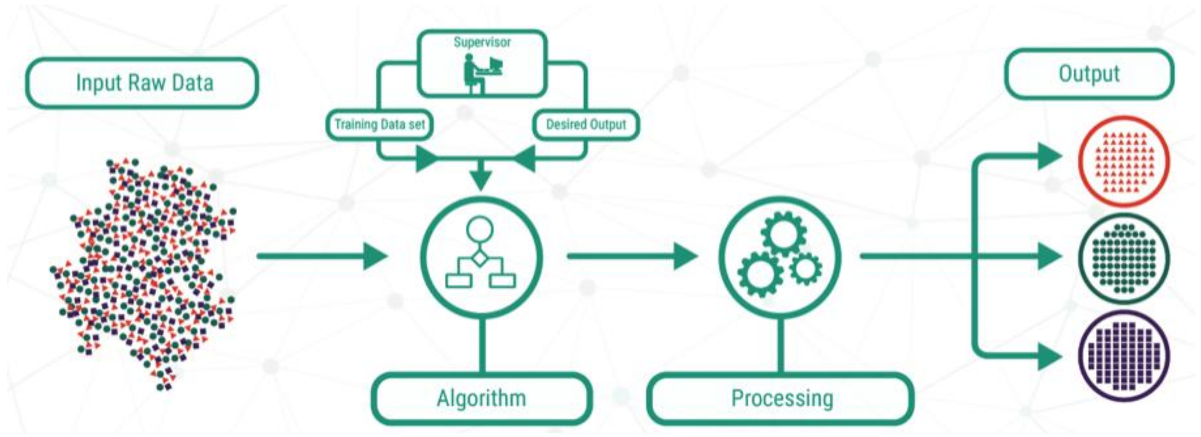


Figure 6: Supervised Learning [39]

A few, most widely-adopted, supervised learning algorithms are:

- Support vector machines, which represent each data as a point in space and attempt to find the widest gap to separate the space into two categories with each data point falling in their correct category. New data are mapped into the same space and according to which side of the divider plane they fall on, their category is determined.

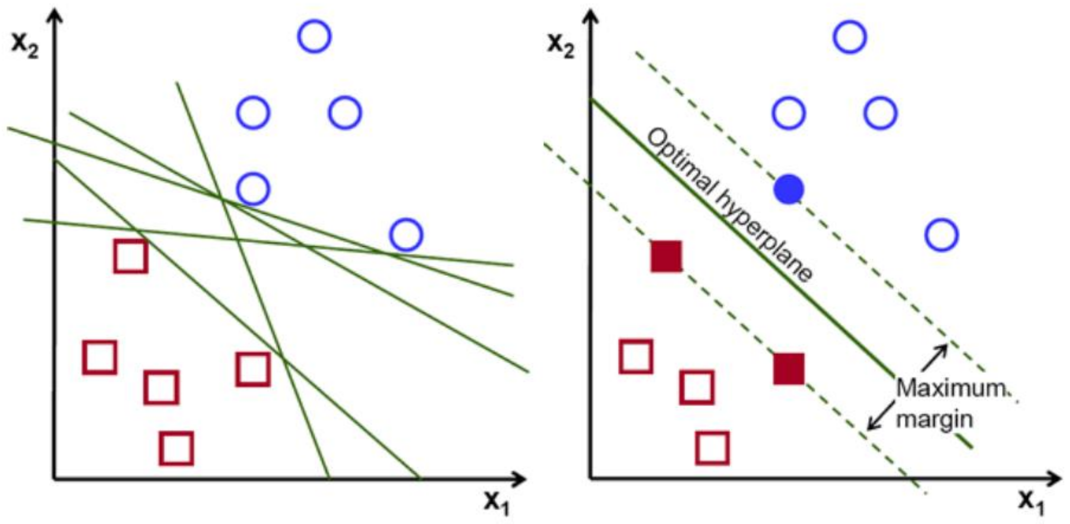


Figure 7: Finding the optimal plane to separate the 2 categories [40]

- K-nearest neighbors, which determines the label of new data based on the labels of the data points located near it in space (neighbors). In other words, the algorithm assumes that similar data exist near the new data and using that similarity, the algorithm deducts the label of the new data. The k parameter determines how many of the nearest neighbors should be taken into account to make that deduction.

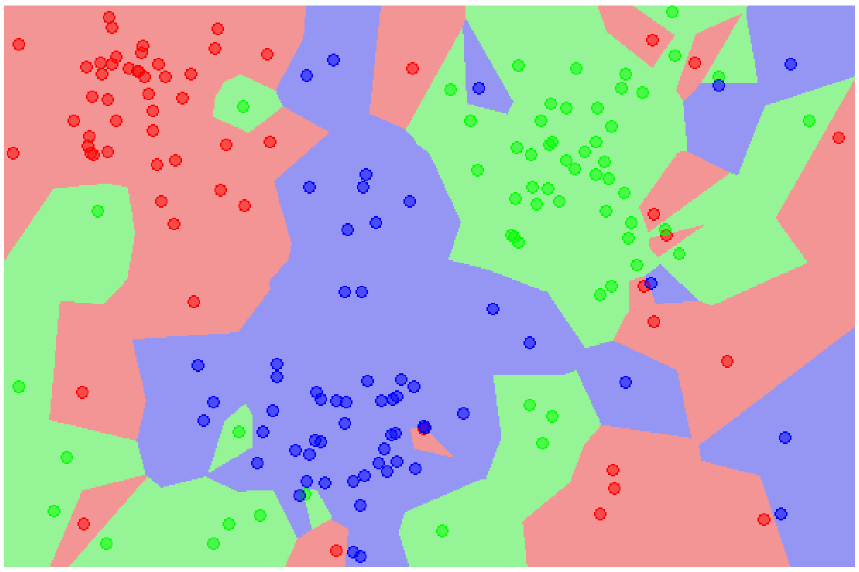


Figure 8: The dataset in a 1NN classification map [41]

- Decision trees, which consist of a model of decisions and their possible outcomes. They are often represented by a series of if-statements used in programming. Decision trees use a flowchart structure, where each internal node is a condition checking an attribute of the input, each branch is the outcome of the condition and each leaf node represents the deducted label. The path taken from root to leaf corresponds to the rules used in the decision-making.

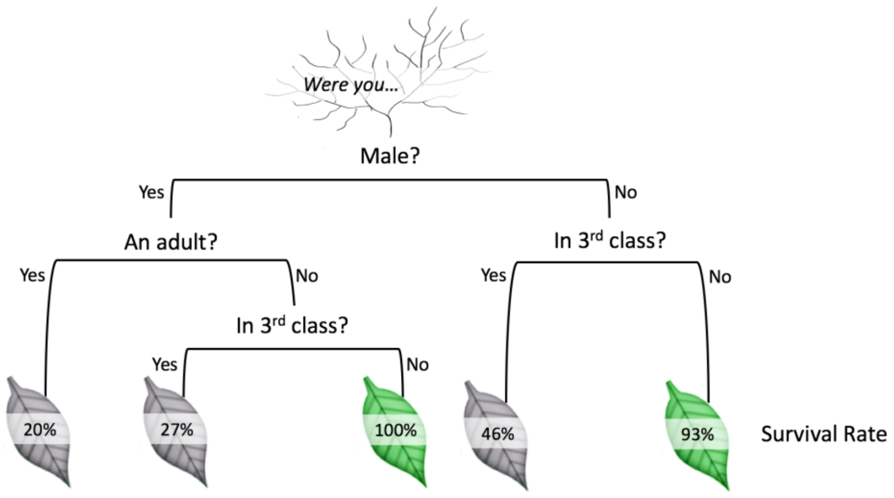


Figure 9: A decision tree [42]

Unsupervised Learning

In unsupervised learning, the algorithm is fed with unlabeled data. This means that it does not know the correct "answer" from the beginning. The algorithm will investigate the input data and strive to discover a hidden structure within the input data. In other words, it identifies similarities in the data and based on the presence or absence of those it processes the new data. To give an example, this type of learning can be used to identify various groups of users with similar interests in order to be subject to the advertisement of a specific product.

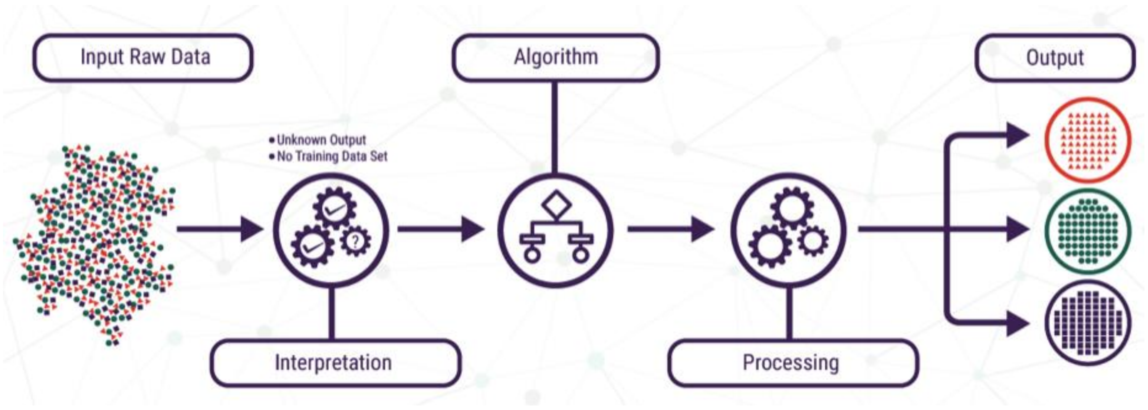


Figure 10: Unsupervised Learning [39]

A few, most widely-adopted, unsupervised learning algorithms are:

- Cluster analysis. It is the task of attempting to segment the data into groups of objects. Objects in the same cluster have similar attributes than objects in other clusters. Cluster analysis itself is not a specific algorithm, rather the general task. There are various algorithms that perform this task, each having its own understanding of what a cluster should be and how to find one efficiently.

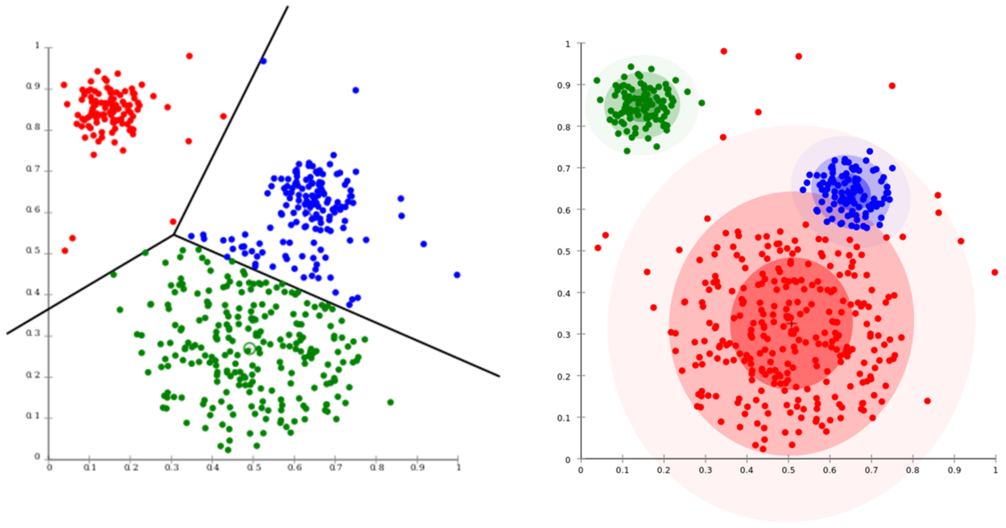


Figure 11: K-means clustering (left) and Distribution-based clustering (right) [43]

- Anomaly Detection, also known as outlier detection. This algorithm is used to detect non-common occurrences in data. These detections are of great use in many fields. For example, an unusual usage of network bandwidth might be indicative of an intrusion, a transaction of an unusual sum of money might mean fraud. System health monitoring also benefit from detecting uncommon data from patients.

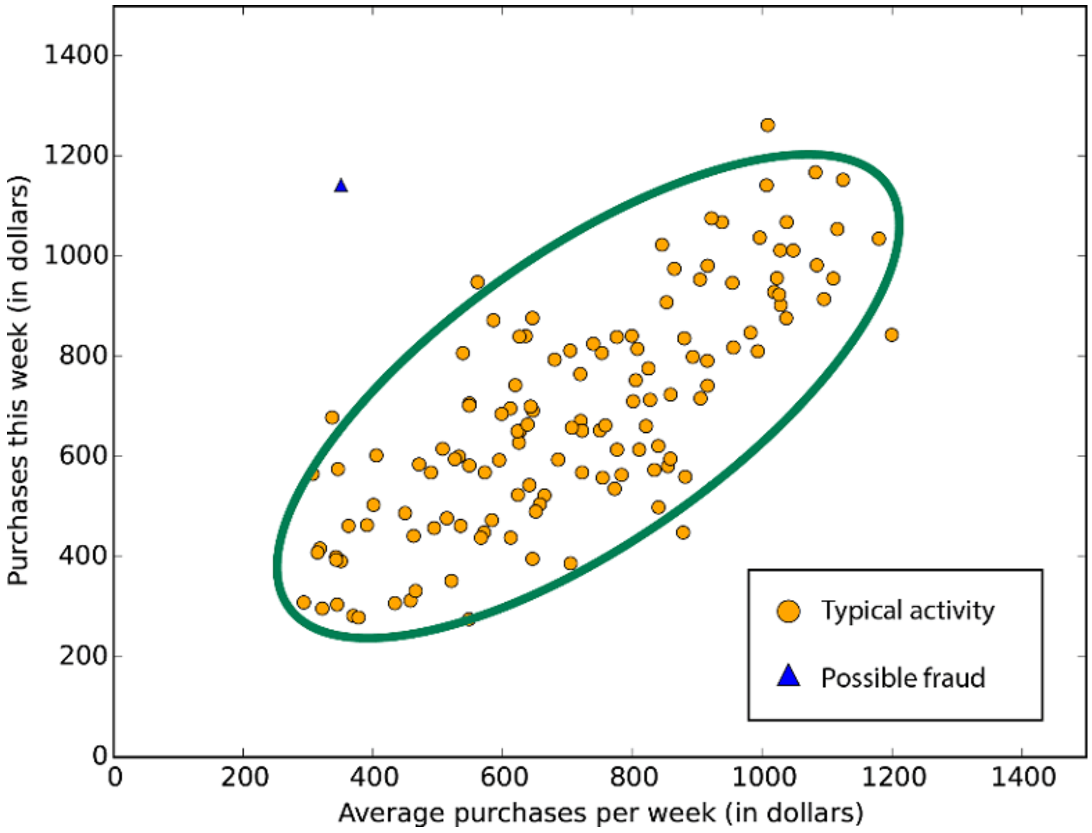


Figure 12: Anomaly Detection of a fraud [44]

Semi-supervised Learning

Semi-supervised learning is a hybrid of the two previous learning approaches. The algorithms here are fed both labeled data and unlabeled data. Usually, the amount of the unlabeled data is far superior than that of the labeled data. This happens because the acquisition of labeled data is costly, especially when there's a lot of data, since it requires a skilled developer to label every single data point, which requires effort and time. Semi-supervised learning can attain improved learning accuracy compared to supervised learning, which would use less but labeled-only data. An example of a semi-supervised learning task is that of a course exam. The labeled data would be a few exercises that were solved during class. The teacher has also provided a lot of unsolved exercises, which the students are to solve on their own to perform better on their exam [45].

Reinforcement Learning

Reinforcement learning consists of a learning approach that closely interacts with its surrounding environment. It interacts with it by performing actions and by trial and error, it learns which action would produce the best possible yields. Reinforcement learning revolves around the agent, which is an object in the scene, and the scene itself. In every iteration, the agent according to the state it is in and the scene, which determines its available actions, decides the best action that would yield the best reward. This brings the agent into a new state/iteration, where the agent repeats the same process having also past experiences as its asset. The agent's (or rather the algorithm's) objective is to achieve the maximum possible reward throughout its actions over a specific duration in time.

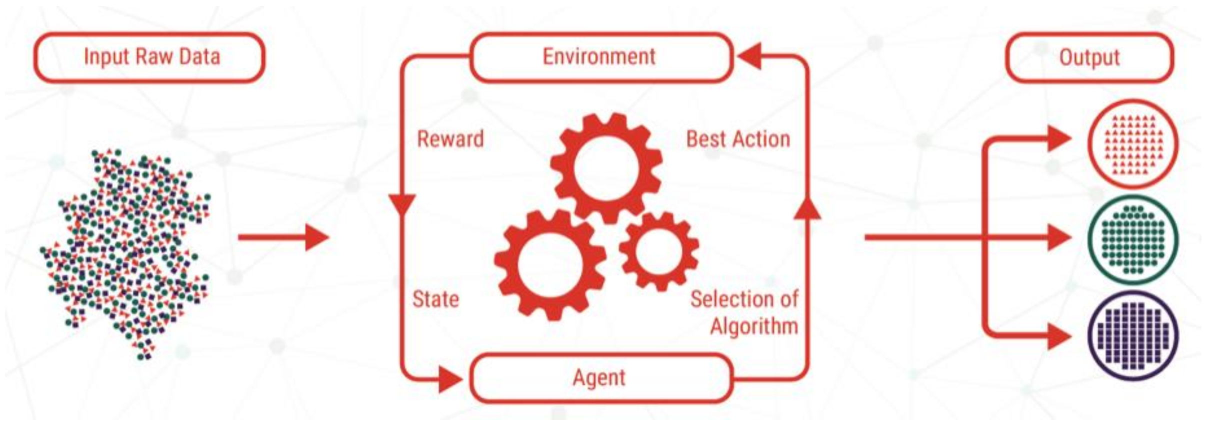


Figure 13: Reinforcement Learning [39]

Reinforcement learning is very suitable for:

- Robotics. An example is to make a robot perform certain type of moves, e.g. running, walking, grabbing items. Through the use of reinforcement learning, prosthetic limbs could vastly help an amputee by recognizing his/her patterns and automatically adjusting the motor movements to attain a more stable and natural movement for the user as time passes.
- Games. For example, let's say that the agent needs is inside a virtual maze with various goodies, each giving a different (negative or positive) reward. The ultimate reward would be for the agent to get out of the maze. After the agent explores the maze a bit to gather information about the environment, it might find a location with goods that give positive rewards. In every iteration of the learning process, the agent could repeatedly go to the same location to keep stacking up the small rewards but that would not be the best long-term reward, which is to escape the maze. In some iterations, the agent will take other routes to explore the rest of the maze to gather more info in order to choose the highest-yielding action. Eventually, it will reach the goal. It's important to note that the best reward might not be immediate, but the agent acts based on the best long-term one.

- Navigation. Self-driving cars have become quite common. Instead of hard-coding what the car should do in each situation, the agent can learn the best possible action in every environment, taking into account safety, ride time, car pollution and passenger comfort.

Artificial Neural Networks

Artificial Neural Networks (ANNs) are an interconnected group of nodes, each being an artificial neuron (inspired by the neurons of the biological brain). ANNs learn by reviewing examples, without being programmed with specific rules. They consist of three main parts: the input layer, the hidden layers and the output layer. The input layer consists of the data being fed to the algorithm and the output layer is the prediction of the label of the input. The middle section (the hidden layers) have nodes, each being connected to all nodes in the previous and the next layer. Each node computes the sum of its inputs by some non-linear function and the result propagates to the next nodes, ultimately reaching the output. In the learning process, the output is compared to the correct output and the deviation is used to adjust the coefficients of the nodes and the connections between the nodes in an attempt to decrease the error in the next iteration. Ultimately, the output layer will reach a point that it represents a result close enough to the right one. At this stage, the learning process can be stopped and a snapshot of the hidden layers is taken, which consists of the model. This model is then used to predict the result of new data based on what the model had learned from the training data.

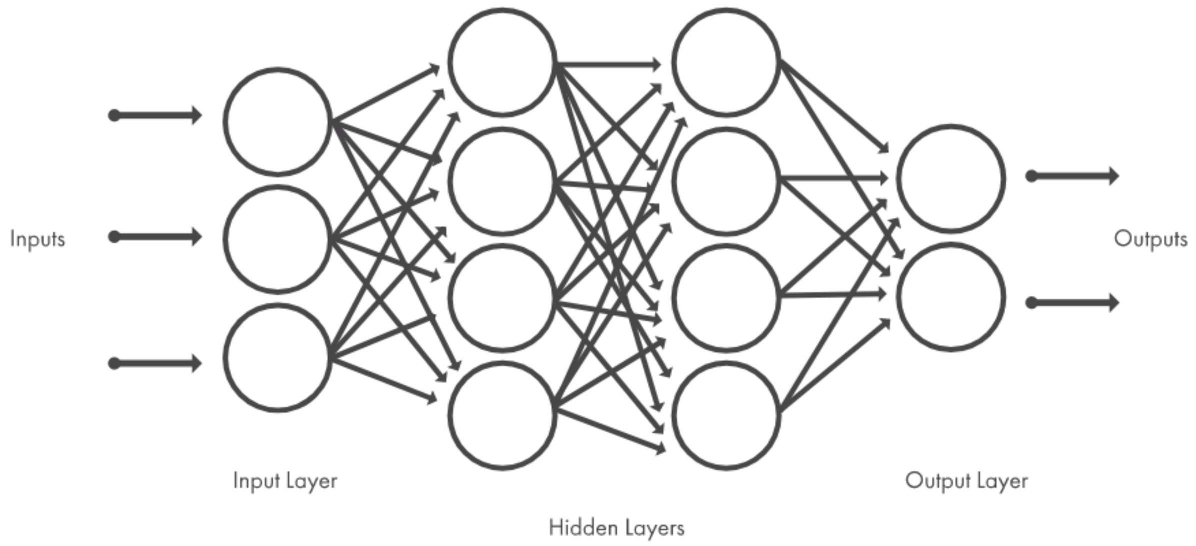


Figure 14: An artificial neural network [46]

ANNs were made to solve problems like the human brain would but in the course of years, their use-case has shifted to executing specific tasks. As such, based on the task, artificial neural networks can belong to any of the above machine learning approaches (supervised, unsupervised, semi-supervised and reinforcement).

Deep Learning

Deep Learning is considered a subset of machine learning. What makes deep learning different from other approaches is that the models learn from vast amounts of data using a multi-layered neural network architecture. With deep learning, there have been astonishing results in the last few years, sometimes even exceeding human-level performance in tasks like classifying objects in images.

Although deep learning was theorized many years ago, only recently it has managed to become useful. This is the case for two reasons:

- It requires a vast amount of information as input. For example, for the development of a self-driving car the model will need a lot of labeled captured images and recorded videos to achieve a sufficient accuracy.
- Because a lot of data is processed in the learning process, there's a high demand in computing power. Here, GPUs come into play. Their architecture allows them to parallelize tasks and with the conjunction of cloud computing, the learning process of a deep neural network could be reduced from weeks to hours.

Deep learning has countless applications in the industry, such as:

- Smart homes. Home assistance devices have become incredibly popular. They can detect patterns in everyday tasks and switch on-off devices throughout the home at the appropriate time.
- Automated driving for detecting traffic light, stop signs, road lanes, even pedestrians in order to avoid accidents.
- Medical devices to identify cancerous cells in an early stage.
- Security systems for authenticating workers at the time of their entry/exit by using image recognition.
- Commercial digital assistants. All such assistants use some form of automatic speech recognition to detect when the user wants to indicate that they're about to speak to the device by using an activation word/phrase. Verbal text, that follows the activation, is also part of a speech recognition process.
- Industrial automation, where the safety of the workers around heavy machinery is a major concern. The trained model can detect whether a worker is within a safe distance to continue operation.

Neural network architectures are usually used in deep learning and thus, they are often called "deep neural networks". "Deep" simply means that the neural networks can have many hidden layers. Normal neural networks have a few hidden layers, but deep ones have tens and can even reach up to hundreds of them, which justifies why it can "consume" a massive amount of data to get to the desired high accuracy.

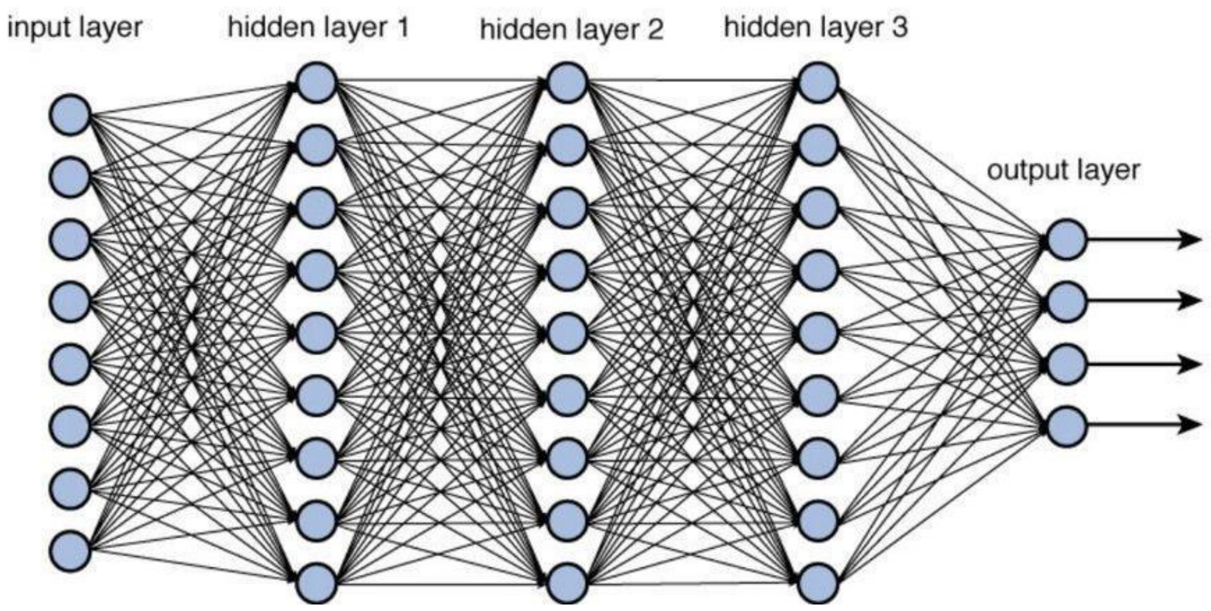


Figure 15: A deep neural network with few hidden layers [47]

Convolutional neural networks (CNN) are one of the most popular deep neural networks. They are widely used to perform classification tasks using as input images, videos, text and sound. They are especially good at processing 2D data, which is why finding patterns in images to recognize objects is a common use-case. Their core task is extracting features from the input images. With the feature extraction, deep learning models can achieve high accuracy for tasks regarding computer vision.

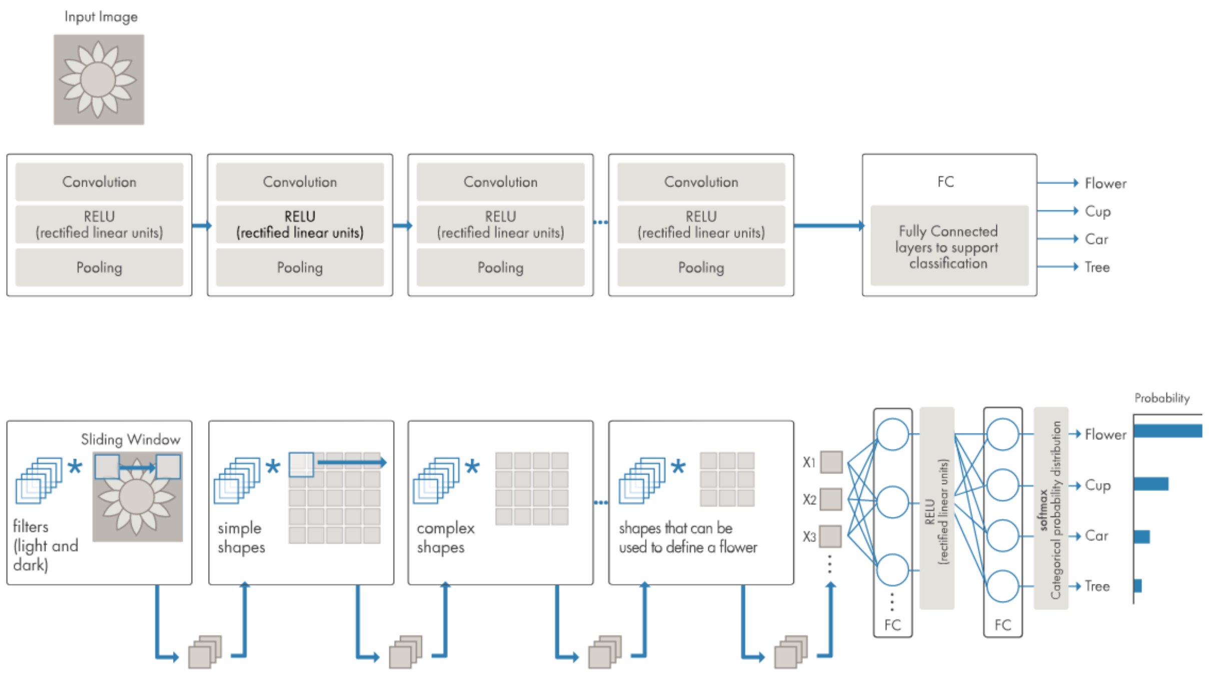

Input Image
Convolution
RELU (rectified linear units)
Pooling
FC
Fully Connected layers to support classification
Flower
Cup
Car
Tree
Sliding Window
filters (light and dark)
simple shapes
complex shapes
shapes that can be used to define a flower
X1
X2
X3
FC
RELU (rectified linear units)
softmax Categorical probability distribution
Probability

Figure 16: A neural network with repeating convolutional layers [46]

The way a CNN works is the following: Firstly, it receives the image in the input layer. In the hidden layers, it alters its copy of data in order to learn various features of the image. Common layers of the:

- Convolution, where various filters are applied to the image in order to "activate" features of the image (e.g. shapes, colors, orientation).
- Activation (ReLU – Rectified Linear Unit), where negative values are mapped to zero and maintaining the positive values. This is an optimization step to allow faster training since negative values (now mapped to zero) won't be carried over to the next stage as they provide no additional info for the certain feature.
- Pooling, where the input is down-sampled in order to reduce the amount of parameters for the next stage. This is also an optimization step as it decreases the number of parameters the networks has to learn.

After the image is processed through many convolutional layers (each learning more complex features than the previous layer), the network reaches the fully-connected (FC) layer, where a vector is produced with size equal to the number of classes the network is able to predict for the image. Each element of the vector contains the probability of the image representing that class. Based on how high those percentages are and using a threshold percentage, the network can classify what object(s) the image contains.

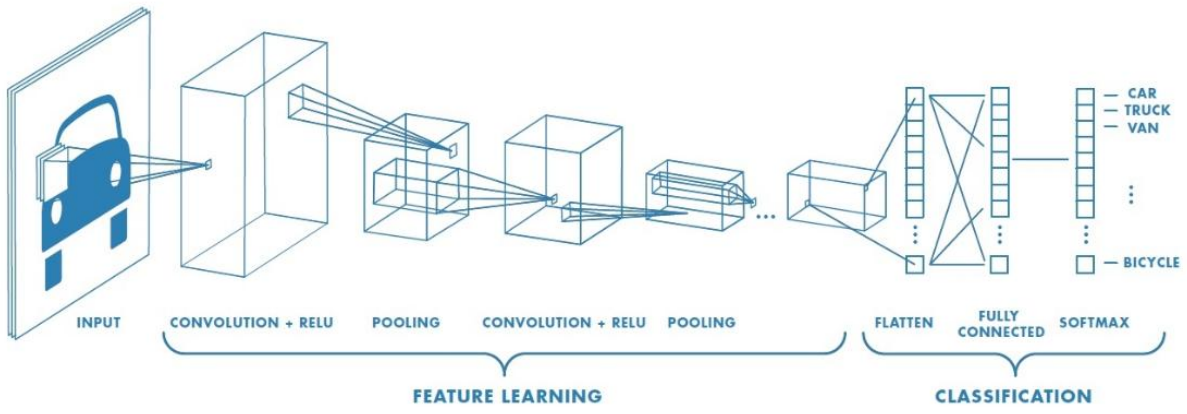


Figure 17: Each training image goes through filters at various resolutions and the convolved output becomes the input of the next layer [48]

There are 2 major differences in the workflow of machine learning and deep learning:

- The most major difference between machine learning and deep learning is how the features of data are extracted. In machine learning, the features are being extracted manually and then, those are being used as input for a chosen classifier to predict the output. In deep learning, the feature extraction and modeling processing are performed automatically and thus, end-to-end learning is achieved.

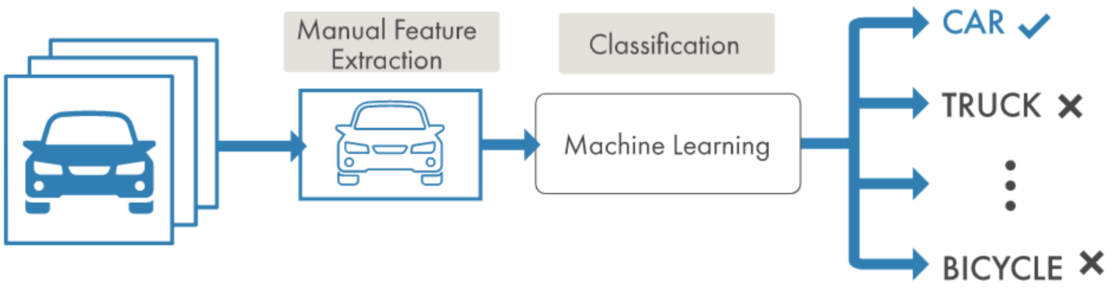


Figure 18: The machine learning approach [46]

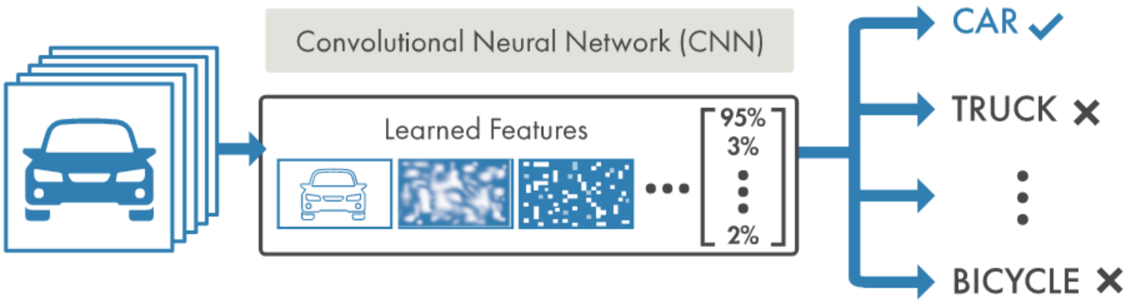


Figure 19: The end-to-end learning approach of deep learning [46]

- Another advantage of deep learning is that it can scale with data. This means that the model can continuously improve with more training data, in contrast to common machine learning algorithms that after some point the model does not gain increased accuracy, even if more data is supplied.

There are three common ways to start training deep learning models are the following:

- Training from scratch, where the developer has to gather a massive amount of labeled data in order train the model from zero. This is used less commonly as it can easily take days, or even weeks, to train the model depending on the amount of data and the performance of the hardware (GPU specifically). However, it does pay off for new applications.
- Transfer Learning, where the model is not trained from scratch. The developer uses a pre-trained model and feeds it new training data with new classes. This approach not only requires less data but it also can reduce the learning process.
- Feature Extraction, where the deep learning model itself is used as a feature extractor. Since all the hidden layers of a deep neural network are used to learn features from the input, these features can be retrieved during the learning process and used as input in typical machine learning models to compensate for their lack of automatic feature extraction. This approach is less commonly used.

Transfer Learning

Transfer learning consists of a learning approach, in which a model that has been trained for a specific task previously is used to train a model for another one (often being similar to the previous one). The fine-tuning process of the model is faster and easier. Transfer learning is often used when aiming for tasks, such as object detection, image classification, speech recognition.

Although training from scratch can have its own advantages, transfer learning has the following benefits:

- It requires less training data than training from scratch. This is because the starting model already has been trained for a similar task and thus, has some "knowledge" on the features of the data it is about to be trained on. Furthermore, there are publically-available pre-trained models, which were trained by other developers (or even companies) on other, and usually large, datasets, which the community can utilize freely. However, if for a specific task a pre-existing model is not suitable, then the developer is forced to train a large amount of data in order to reach a respectable accuracy with training from scratch.
- It reduces training time and consequently, computing resources. As mentioned by the previous reason, the new training data can be less, which is the reason that the learning process is smaller.
- It can improve the performance and the accuracy of the model compared to models that were trained from scratch as the algorithm receives more varied training data and thus, the model is in a position to "understand" better what the new data can (or cannot) be classified as.

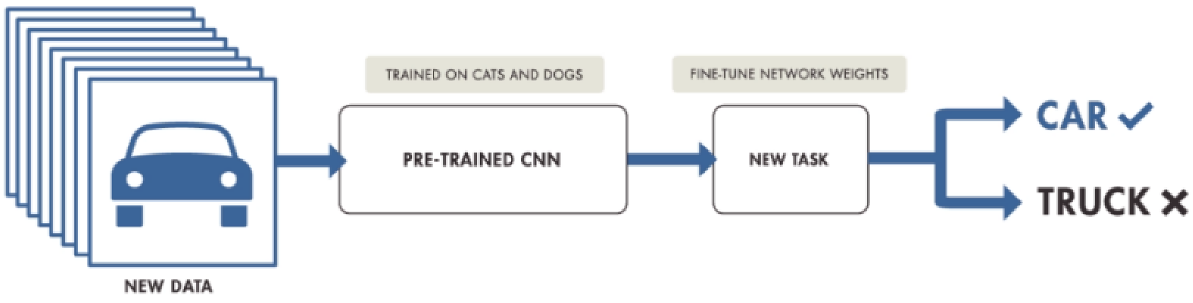


Figure 20: The new task receiving new data and a pre-trained network in the input to transfer-learn the features [49]

The way transfer learning works is:

1. The pre-trained model is loaded. That model could have been trained on millions of datapoints knowing thousands of classes. The early hidden layers have extracted simple features, such as simple shapes (lines, edges) and colors. The last layers contain the high-level features, such as the outline of a car, which are task-specific.

2. Since the pre-trained model was trained on similar but specific tasks, some modifications are needed. As the last layers are the ones that contain the task-specific features, those are the ones that need to be adjusted to tailor the model to the needs of the new application. This way the new model already knows how to extract low-level features and thus, the learning process is reduced.
3. At this stage, the network is trained on the new training data. The training can focus on the classification layers for the extraction of the high-level features. The training datapoints can be in the hundreds, but with the experience of the millions of datapoints, the network will not lack in performance.
4. Lastly, the newly trained model is obtained and through some validation data, the accuracy of the network can be assessed. If it's not high enough, further learning is required in order to improve it.

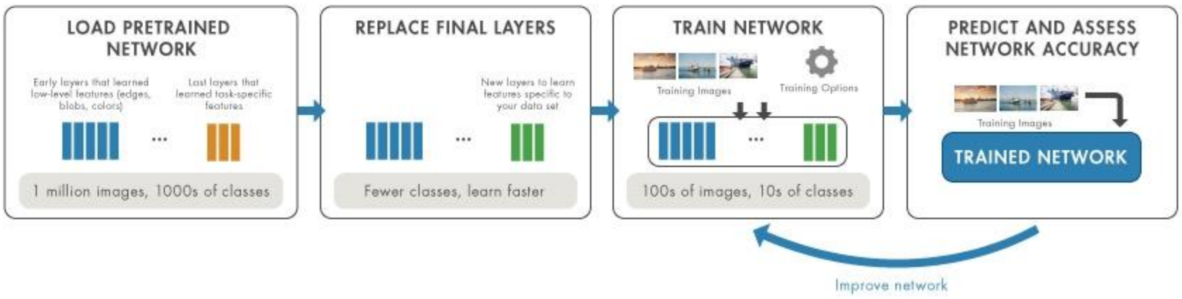

LOAD PRETRAINED NETWORK
Early layers that learned low-level features (edges, blobs, colors)
Last layers that learned task-specific features
1 million images, 1000s of classes
REPLACE FINAL LAYERS
New layers to learn features specific to your data set
Fewer classes, learn faster
TRAIN NETWORK
Training Images
Training Options
100s of images, 10s of classes
PREDICT AND ASSESS NETWORK ACCURACY
Training Images
TRAINED NETWORK
Improve network

Figure 21: The workflow of transfer learning [49]

As shown in the below graph, compared to a model that hasn't implemented transfer learning, the performance of a neural network with transfer learning applied has a "head start" since a pre-trained model is used, the learning/training time is shorter due to the smaller training data size needed and the final accuracy is higher. As such, whenever there is a pre-trained network for a similar task available, it should be taken advantage of. It saves time, effort, resources and the end-product can be very performant.

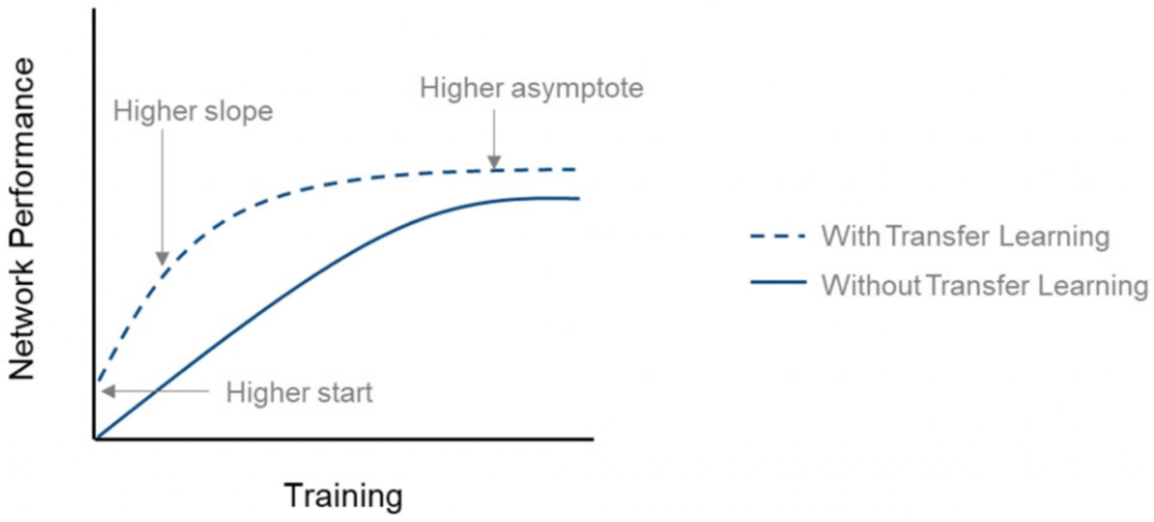


Figure 22: Transfer learning vs Training from scratch [49]

Object Detection

Object detection is a form of computer vision technology that is specialized in locating objects in images and videos. Object detection has many applications, such as disease diagnosis in medical image analysis or pedestrian/lane detection in self driving cars that enhance driving experience taking into account all safety measures.

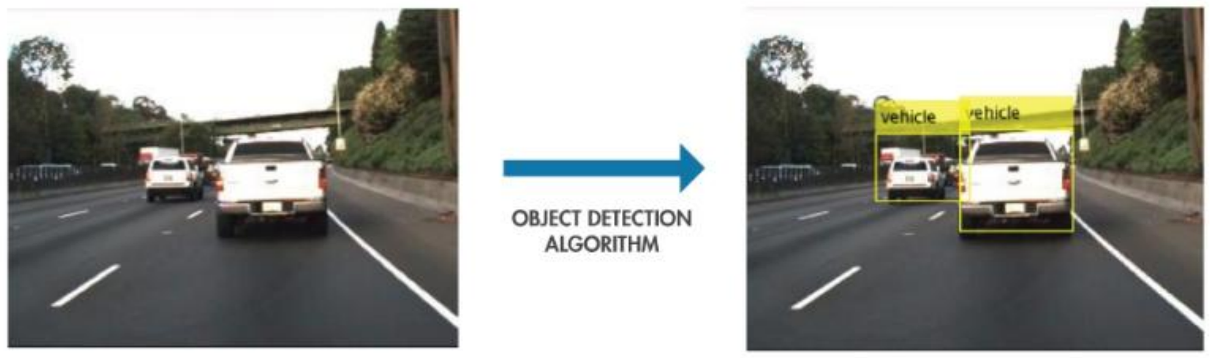


Figure 23: Detecting and locating vehicles in an image [50]

Object detection can be used through various approaches, traditional machine learning or deep learning.

Using Deep Learning

The Convolutional Neural Networks (CNNs) algorithm and many of its variants, such as Region Convolutional Neural Networks (R-CNNs), Fast Region Convolutional Neural Networks (Fast R-CNN) and Faster Region Convolutional Neural Networks (Faster R-CNN) are common for object detection tasks with the deep learning approach.

There are two ways to create an object detection system:

- Creating an object detection model from scratch, which requires a large collection of labeled data to train the network and a lot of time for the learning process itself.
- Making use of a pre-trained object detection model to use transfer learning. This gives the advantage of virtually increasing your own dataset as the pre-trained model has received its training on a massive dataset, which is different than the one it is about be trained on. As mentioned before, this route requires less data and the resulting model is achieved in less time and often with higher accuracy.

One can choose between two types of object detection networks, each having its own workflow with advantages/disadvantages:

- Single-stage networks
- Two-stage networks

Single-stage networks attempt to make the predictions for every region in the image with the use of anchor boxes. Those predictions are, then, decoded to produce the bounding boxes for the predicted boxes [50]. These networks are faster than two-stage networks but they come with the cost of accuracy, especially when the image contains objects that occupy a small number of pixels. Thus, single-stage networks are more suitable for real-time applications.

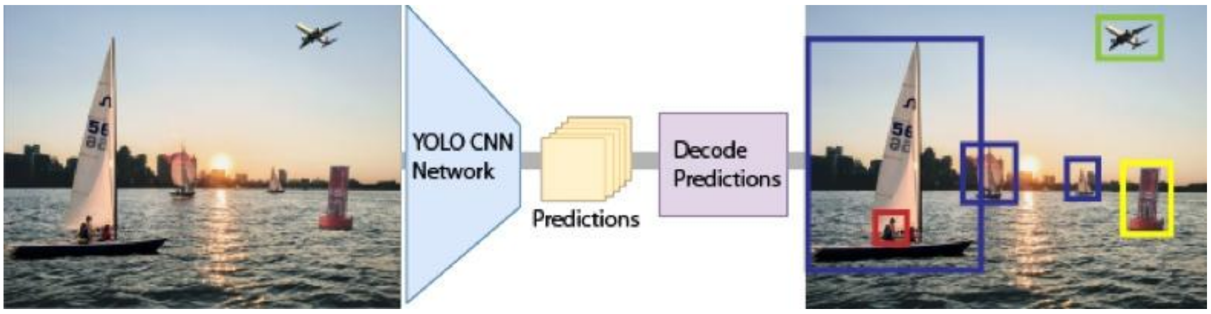


Figure 24: YOLO (single-stage network) object detection workflow [50]

Two-stage networks produce predictions based on two stages. The first stage predicts regions of the image that have good chance of containing one or multiple objects. These predictions are passed to the second stage, where the objects of each region are classified [50]. These networks might be slower than single-stage networks, but are definitely more accurate in their findings. As the inference time is significantly larger, they might be usable in almost real-time applications (by sampling frames of a video feed).

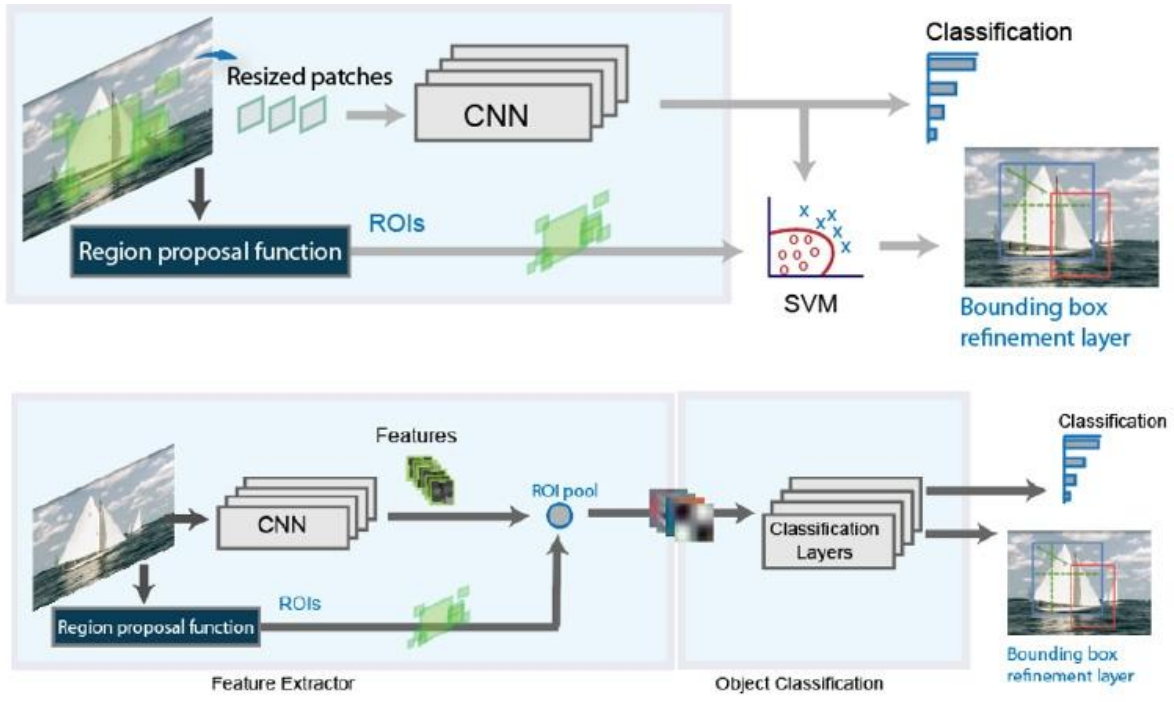


Figure 25: Two-stage network object detection workflows using R-CNN (top) and Fast R-CNN (bottom) [50]

Using Machine Learning

Popular methods of object detection via the traditional machine learning approach are aggregate channel features (ACF) and support vector machines (SVM) using histograms of oriented gradient (HOG) features. These algorithms are often used in applications like vehicle or pedestrian detection.

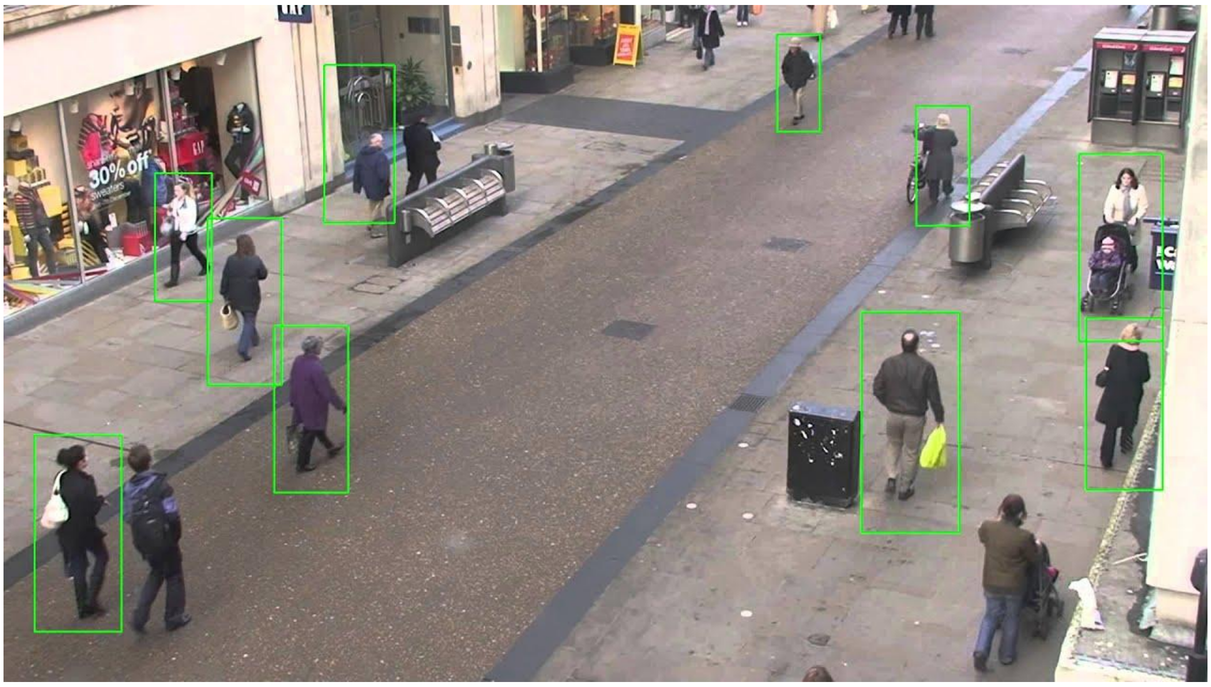


Figure 26: Pedestrian detection with SVM using HOG features [51]

Other approaches

Object detection can also be achieved via other techniques, aside from machine learning. One that is used quite often in mobile devices is feature extraction paired with Random Sample Consensus (RANSAC). Specifically, feature extraction detects interesting regions of an image and stores that information into a feature vector. This process is essential for reducing the size of information an image has, especially in large ones, where the pixels might be in the millions. For example, a white wall in a large image could be represented by the location of its 4 edges instead, dramatically reducing the information needed to store. This makes tasks after the feature extraction, like RANSAC, perform faster as they will have to process a smaller quantity of data. RANSAC is the iterative process of approximating the parameters of a mathematical model from a data set containing outliers. RANSAC will try to identify outliers in the feature vector of the previous stage and estimate the appropriate parameters of a model that does not contain the detected outliers. This process is useful in detecting the location of an object in an image that matches specific features. In the following example, there is a visual of example of how a view of a box is detected and located in a scene containing that box:

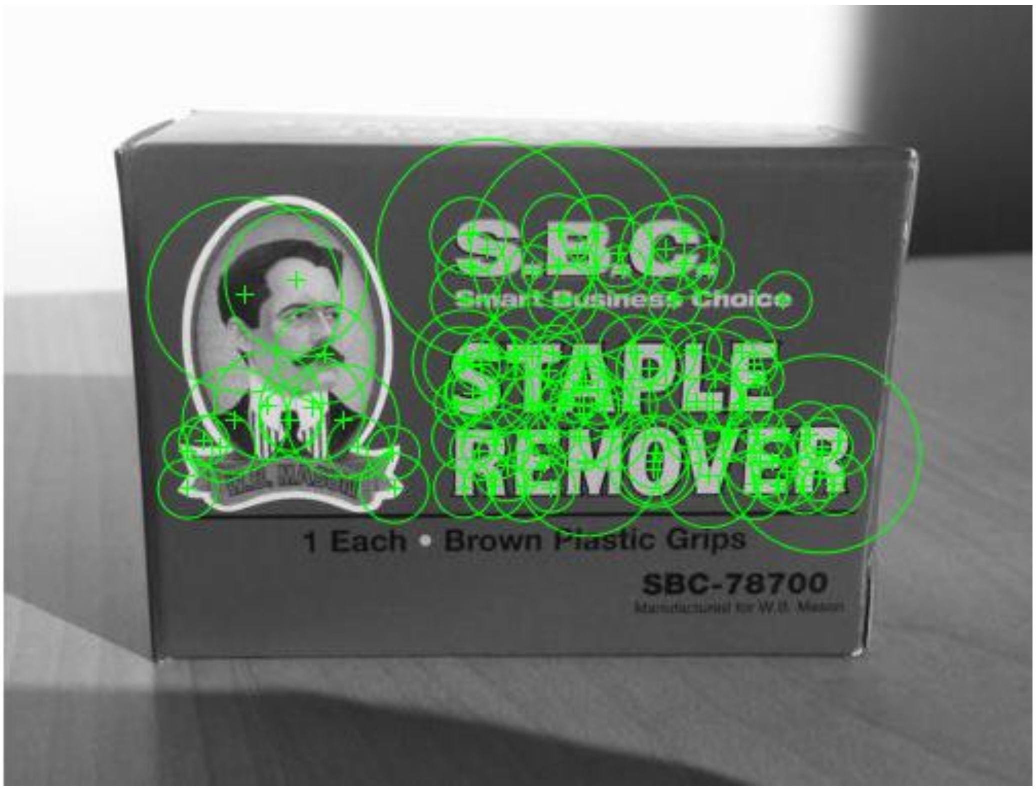


Figure 27: Extracting the most notable features in the box image [52]

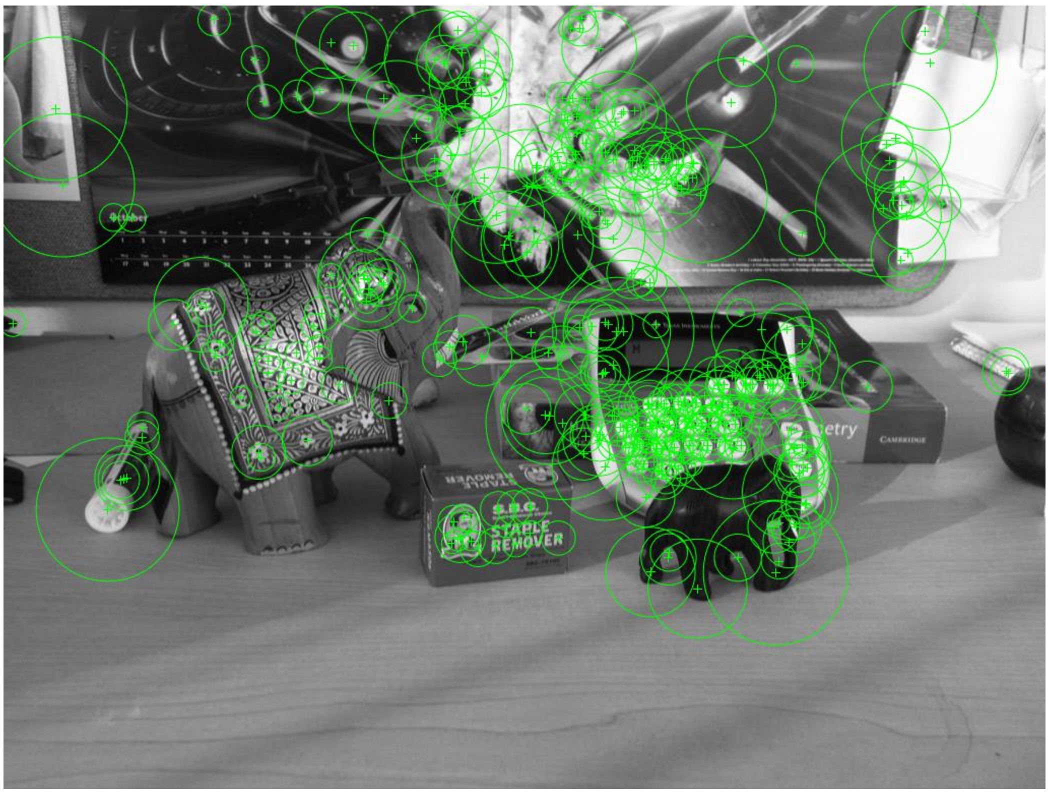


Figure 28: Extracting the most notable features in the scene image [52]

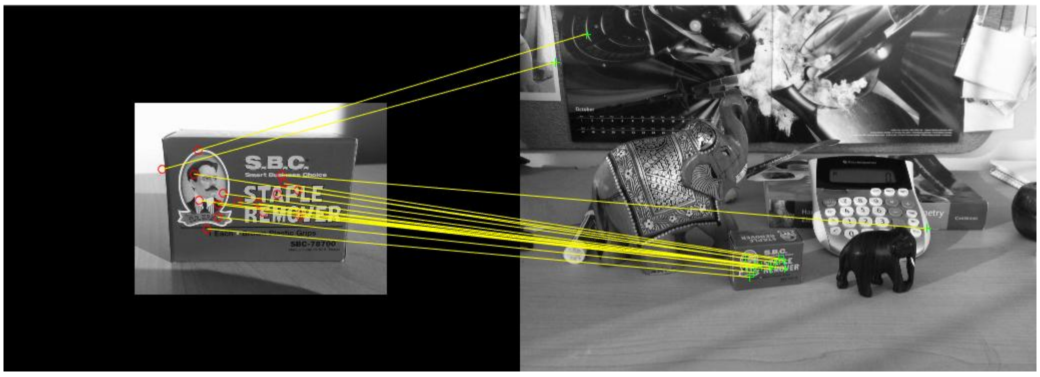


Figure 29: Match the features of the box image to the ones in the scene image [52]

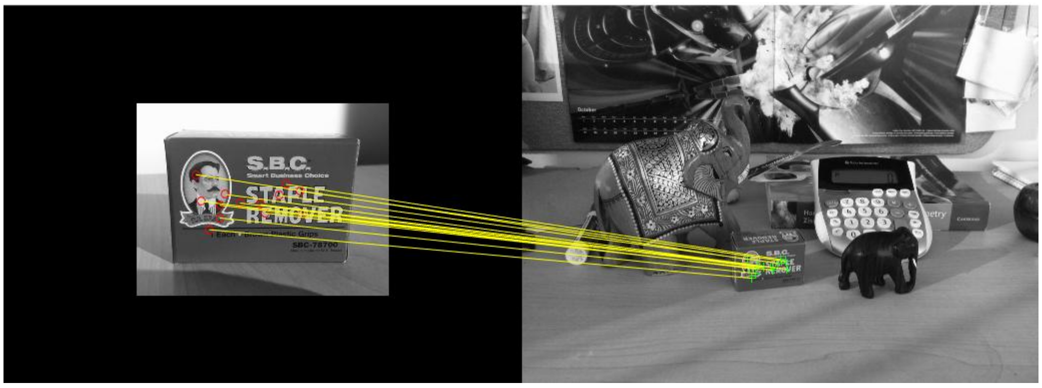


Figure 30: Remove the outliers using RANSAC [52]

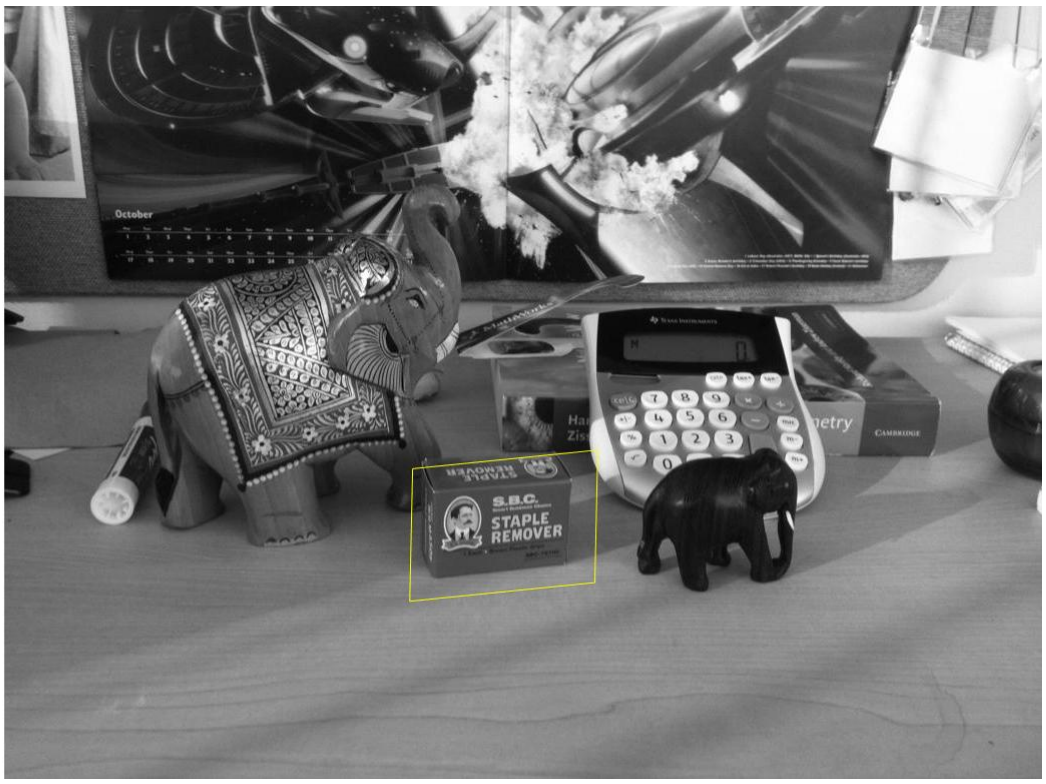


Figure 31: Locate the box image inside the scene image [52]

Enabling Framework

There are numerous machine learning frameworks in the software industry. By using these, developers are able to develop faster and easier. The frameworks provide pre-build components with customizable settings/parameters so the developer does not have to implement himself/herself, making model training easy. All these tools usually provide an intuitive UI and documentation, which the developer can refer to in order to effectively use and tweak the appropriate modules to tailor the application exactly to the specified needs. As the machine learning field is constantly evolving, most frameworks are open source platforms, which makes contribution of researchers much easier. Here is a list of the most widely-used frameworks adopted for both research and commercial purposes:

TensorFlow

Tensorflow is maintained by Google and is one of the most popular frameworks as it has a massive community support, as well as a variety of in-build features. It supports regressions, classifications and neural networks among many other tasks. This framework can be installed on CPU or GPU, with the latter being the optimal choice as it is more efficient with the extra computing power, especially when using the deep learning approach. Tensorflow is an open-source library, which makes it an excellent choice for researchers.

K Keras

Keras is a framework developed by a Google engineer and is written in the python language. It is known for training deep neural networks and it focuses on a friendly and modular interface. This framework can be used in conjunction with other ML frameworks (such as Tensorflow and Microsoft Cognitive Toolkit) to provide a high-level API for easier deep learning model development. Although this framework might be able to decrease multiple lines of code into a single line, it also means that the environment is less configurable.

Microsoft Cognitive Toolkit (also known as CNTK) is a deep learning framework, just like Keras. It primarily performs convolution of neural systems and it can also be used to create other machine learning models. It provides rapid training and intuitive architecture as it supports multi-machine and multi-GPU backend. This framework is maintained, as its name implies, by Microsoft.

Caffe2 is a deep learning framework written in C++ and is developed by University of California in Berkeley. It offers CPU & GPU acceleration, as well as switching between the two with a single setting. Caffe2 is considered one of the fastest frameworks available as it is able to process 60 million images in a day with a single GPU. Therefore, it is a popular choice for research projects and rapid prototyping.

Chainer is a python-based deep learning framework. Its development started in Japan and thus, has a large support within the Japanese community. Chainer is written on top of Numpy and CuPy libraries and is the first platform to use dynamic architecture. It performs very well on large scale systems, like CPU and GPU data centers.

MXNet is a scalable deep learning framework. It is an open source library with a backend written in C++ and CUDA. One of MXNet’s advantages is its support for a large variety of programming languages. It performs extremely well on multiple machines and GPUs, which is why it is utilized on Amazon’s Web services.

PyTorch is an open source machine learning framework, which is mainly developed by Facebook. It supports a python interface, along with a C++ one. The PyTorch library functions using a dynamically updated graph, which means that changes are allowed to be made by the developer in the learning process. The library offers high-level features, such as tensor computing and deep neural networks, along with many pre-trained models.

Scikit-learn is another open source library using mainly the python programming language. Some core algorithms are written in Cython to achieve better performance. The framework utilizes other common libraries, such as SciPy, Numpy and Matplotlib. It offers algorithms with the supervised and unsupervised approach and is able to implement various modeling techniques, like classification, clustering and regression.

Graffiti Detection Application

Graffiti vandalism

Graffiti vandalism has been a long-term issue created by many youngsters for fun, by adults for propaganda purposes, by gangs for territory marking, by people that deliberately want to destroy/damage someone else's property, etc. The only graffiti type that is not considered as a vandalistic act is that of urban graffiti, as it usually is done by professional artists, which have the owner's consent when drawing on his/her property. All the other types of graffiti are considered illegal in most countries as it's inappropriate to draw on someone else's property without consent. Spaying on a wall might be harmless, but the resulting graffiti might have a negative influence on the people passing by. For example, a customer seeing graffiti on store entrances or commercial vehicles can easily derive to the conclusion that the company has lowered standards, which has a negative impact on the company's reputation. Another example is graffiti on ancient monuments, which destroys the cultural heritage of the country. Graffiti vandalism has also increased the theft rate from stores that sell products that are used for committing graffiti, such as sprays. Gangs and political parties also often resort to using graffiti to get their message across, which often concludes to fatal conflicts with other gangs and parties respectively. Aside from the casualties, the cost of removing graffiti can prove quite substantial over time. It is clear that graffiti vandalism has a negative impact on the surrounding community, which is something that must be dealt with.

Solution

Preventing graffiti vandalism is quite a hard task. Most vandals that commit them act at "quiet" times, when few to no people are around, often being at night. Therefore, community policing is hardly considered a solution to stop the act from taking place. The most obvious solution would be allocate resources to security. Using guards that patrol an area can be effective but it is not enough as someone can easily spray on a wall in a couple of seconds to a few minutes while the patrol is on another area. Tackling the problem with just human resources seems quite impossible, at least without spending a huge amount of cash. However, with the use of an auto detection system that is 24/7 present monitoring an area it is possible. Machine learning is the perfect candidate for performing this task. It can work with the today's infrastructure of security systems as the task of detecting object using machine learning requires just the frames, which the cameras already provide. However, training a machine learning algorithm does demand an initial effort that will be paid off in the long-term. Creating a dataset comprised of images with graffiti annotations is a tedious, but necessary, step as the algorithm needs to learn what "graffiti" means, and even more so when graffiti does not have a specific form. In the scope of this thesis, an attempt will be made to create such a system, which will be evaluated in both accuracy and speed.

Implementation

A real-time application will be made, which will be able to detect and locate graffiti tags in an image with the use of a camera. A setting will be available within the application to store the classified image in memory, in case it's needed for further processing. To develop this mobile app, there is a need for specific set of software and hardware requirements.

Software requirements

TensorFlow and its Object Detection API

For the application, it's decided to use the tensorflow [53] framework. It's well-maintained by Google and has good community support, which will make the development easier and faster. Specifically, a subset of the tensorflow framework will be used, which is the object detection application program interface (API). This API also provides many pre-trained models and configurations to apply transfer learning for even faster development. Python will be the programming language that will be used to interact with the API. In this project, tensorflow's version is 1.12.

Android SDK and Android Studio

As the application will be made for android smartphones (as this is the available testing hardware in this project), the android software development kit (SDK) is required. It is an essential tool, which produces an executable file for android smartphones. Android Studio [54] will be the code editor, in which the source code of a demo camera application, provided already by tensorflow [55], will be edited (using the java language) to suit the graffiti system. Android Studio is also the compiler that will produce the .apk file with the use of the SDK.

Hardware requirements

Performant GPU

Since tensorflow's object detection API uses deep learning under the hood, a high-performance GPU will contribute greatly into achieving a high accuracy in a relative short training time. For this project, there is available a single NVIDIA GeForce GTX TITAN X GPU [56] with a compute capability of 5.2, which will be put to use.

Smartphone with a Camera

Aside from the GPU needed for the training itself, any android smartphone with a camera will do. In this project, a smartphone [57] with Android 6.0 (Marshmallow) installed will be utilized, which is more than enough as the object detection API requires an android version of, at least, 5.0 (Lollipop). Smartphones with better hardware are likely to perform better when it comes to detecting graffiti as close to real-time as possible.

Methology

The methology is quite simple. The first step is to acquire the dataset in order to feed the algorithm with images and their annotations. The creation of the dataset is explained in steps 1 and 2.

1. Collecting a set of images is required to feed the model with as many examples as possible of the object it should be looking for. The more examples, the better, as it makes the algorithm immune to features that are not relevant.
2. Annotating the collected images is necessary to indicate to the algorithm which parts of the each image are containing graffiti tags it is looking for.

The next step is to adjust the data to the needs of the application and the framework used. With the same dataset, but with different application or framework the data might need to be modified differently. These steps are included in the step 3, 4 and 5.

3. Analyzing and adjusting the images is an optimization approach. As images can be quite large and the resources limited, it might be useful to resize images to a resolution that frameworks work with best.
4. Splitting the data into train and test subsets is required in all machine learning approaches. The model trains on the train dataset, while it also needs a dataset to evaluate its errors.
5. Generating tensorflow records is a step needed for using the tensorflow framework. It is a form of adjusting the data to a format that the framework is familiar with.

The next, and most important, step is the learning process of the algorithm in order to make the model to detect graffiti tags with as high accuracy as possible. This approach is coved by steps 6, 7 and 8.

6. Getting a pre-trained model is necessary to apply transfer learning from one pre-trained model to another. A configuration file is needed to tweak the learning process itself.
7. Training the model is the main step of the whole project.
8. Evaluating the trained model is a verification stage, where it is checked through metrics how well the model is performing in the specific task.

Lastly, step 9 and 10 explain how the model is used in a real-time application.

9. Exporting the inference graph is the process of acquiring a snapshot of the model, which is used in the application itself.
10. Integrating the frozen graph into the android application is the final step, where the application itself is tested.

Collecting a set of images

As the first step, a lot of images need to be captured to serve as input for the model. For this study, approximately 2000 images were taken, most having graffiti tags. Many of the images deliberately had drawings that reminded of graffiti tags, in an attempt to “fool” the algorithm and ultimately, achieving greater accuracy. The images were captured with as many variable conditions as possible in order to increase the accuracy of the model detecting the graffiti tags. Some variables that were introduced intentionally are:

- Low-light, where the images were usually captured at night or generally in an environment with low brightness

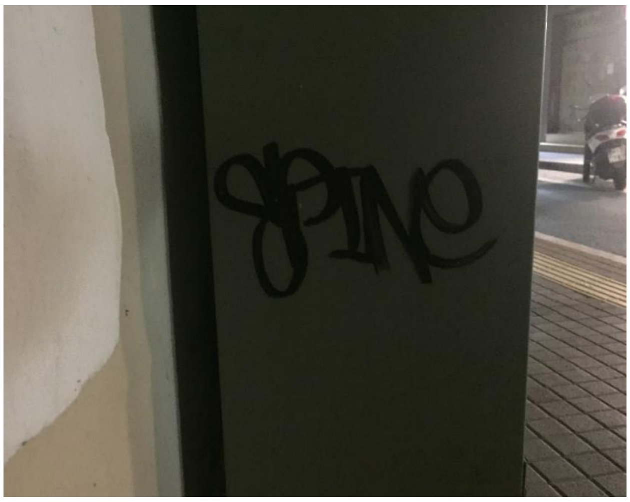


Figure 32: Graffiti tag in a dark-colored surface

- Partially-occluded graffiti tags by other objects, such as cars, fences and trees

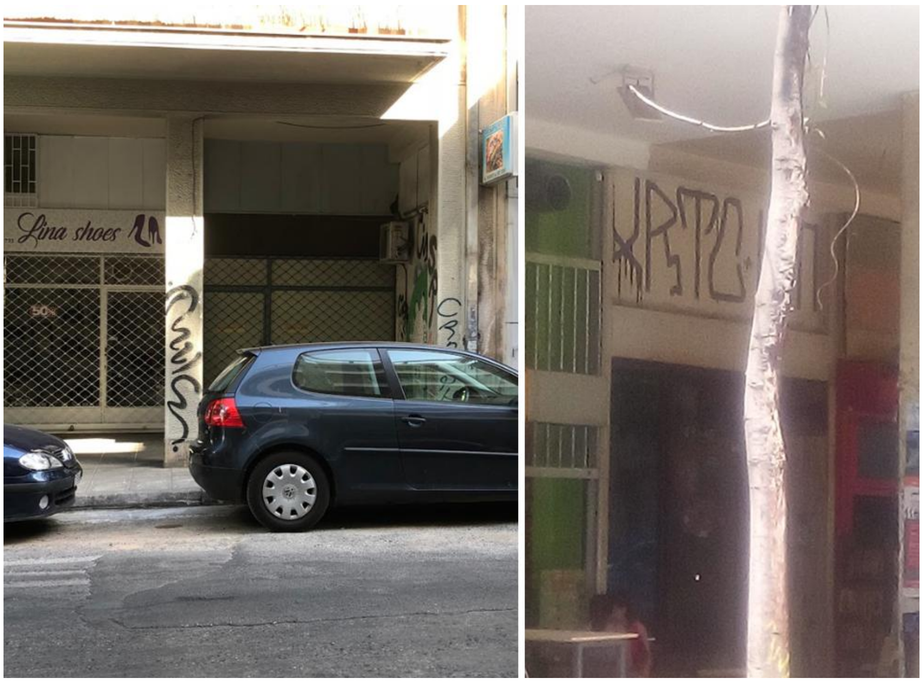


Figure 33: Graffiti tag partially-occluded by car (left) and by tree (right)

- Filters, such as blur

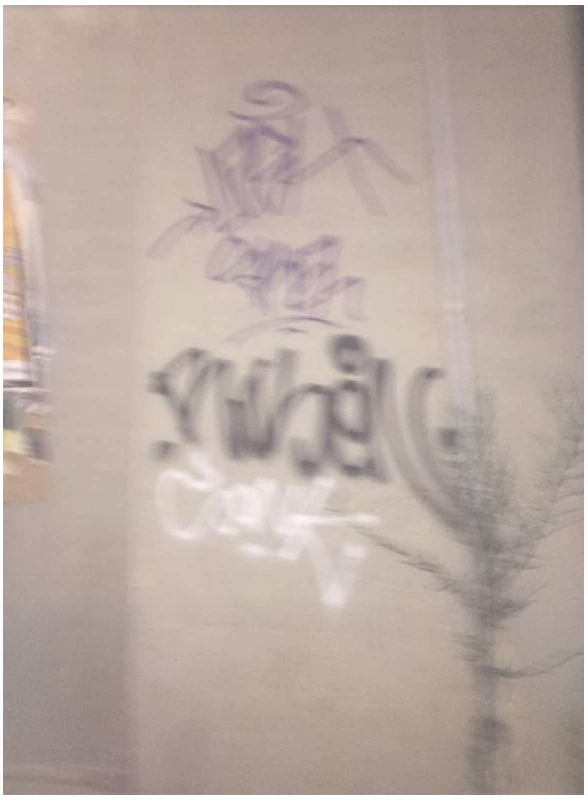


Figure 34: Blurry graffiti tags

- Unique surfaces, that introduced reflections and patterns

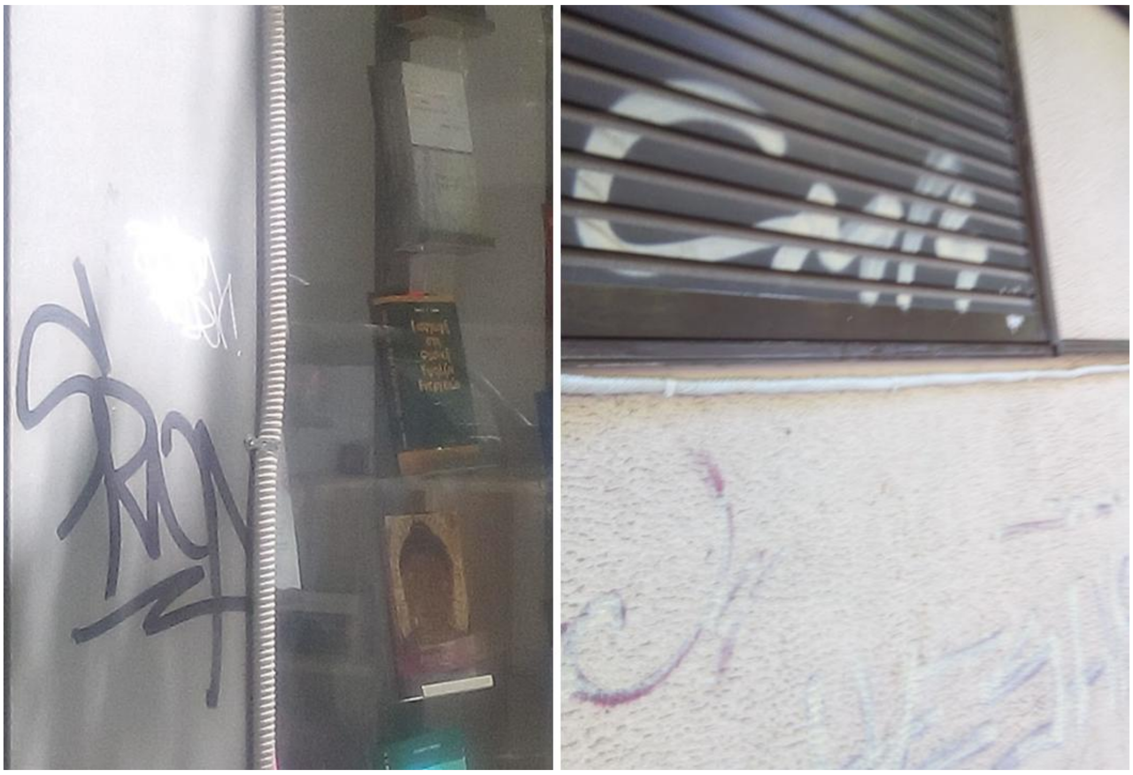


Figure 35: Graffiti tag on reflective surface (left) and on surface introducing a pattern (right)

- Graffiti tags on surfaces containing other information, such as an image or text

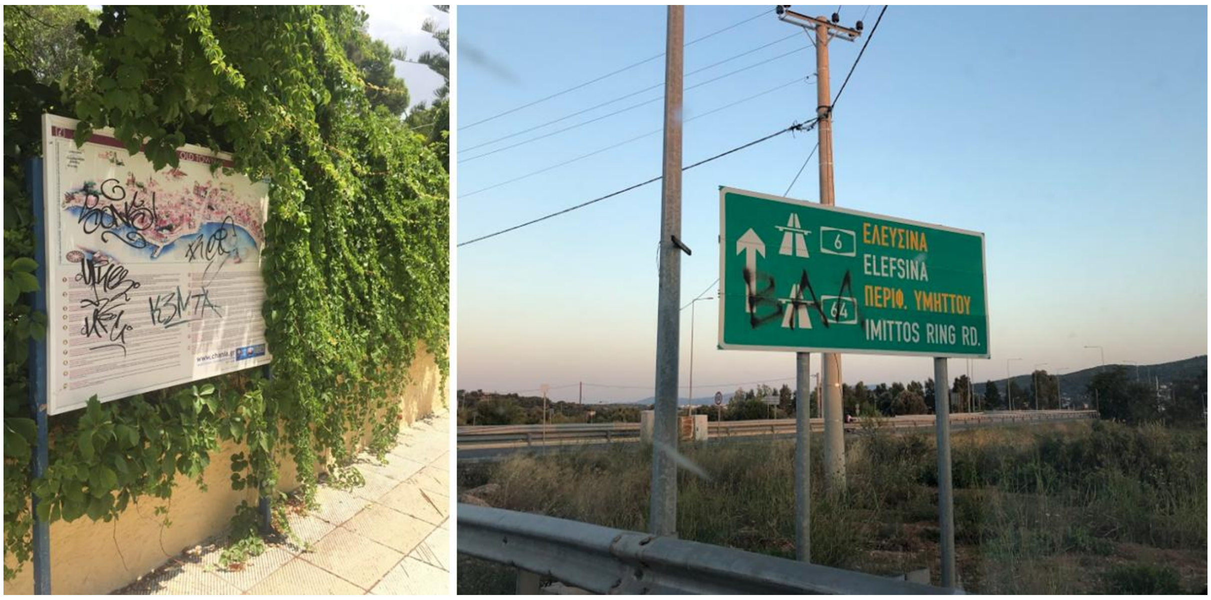


Figure 36: Graffiti tags on images and texts

- Text or shapes on surfaces that aren’t considered as graffiti tags, such as road names, street numbers and shop names

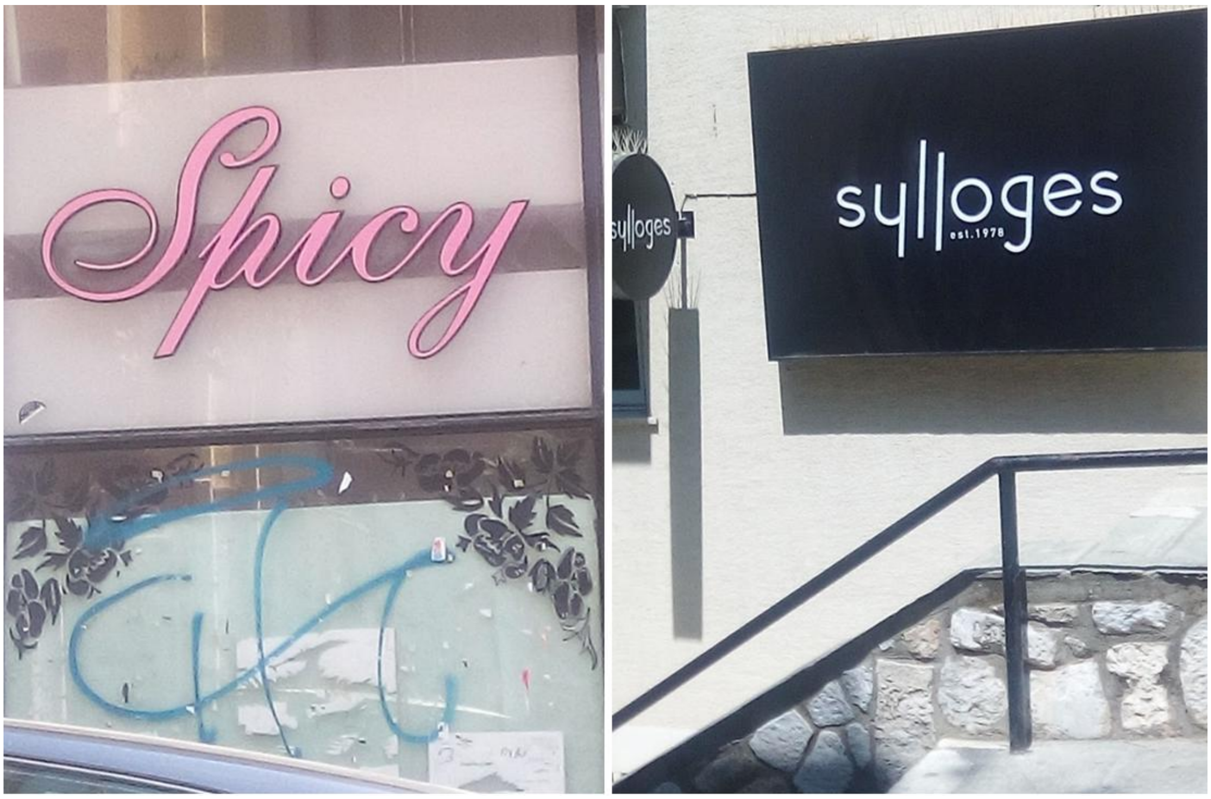


Figure 37: Text that mimicked the style of graffiti tags

- Shadows

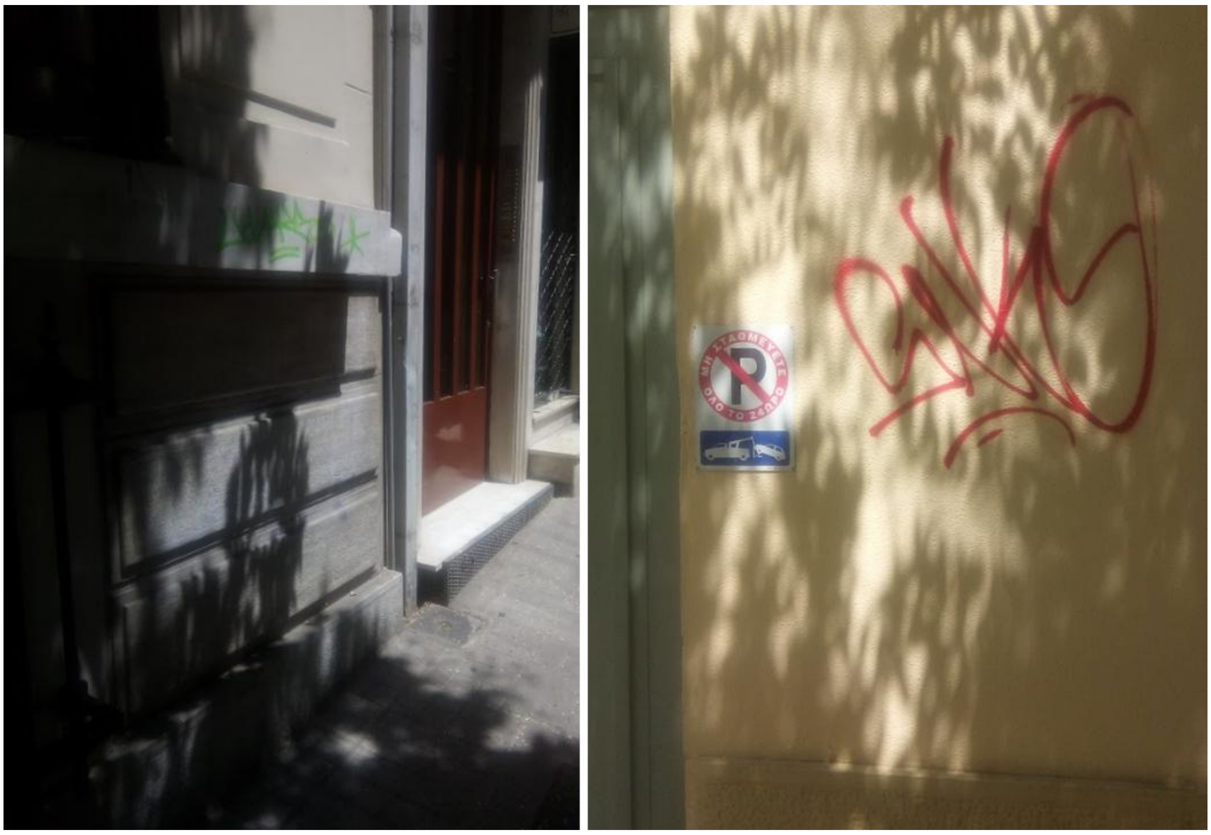


Figure 38: Graffiti tags with patterns introduced from shadows

- Image orientation

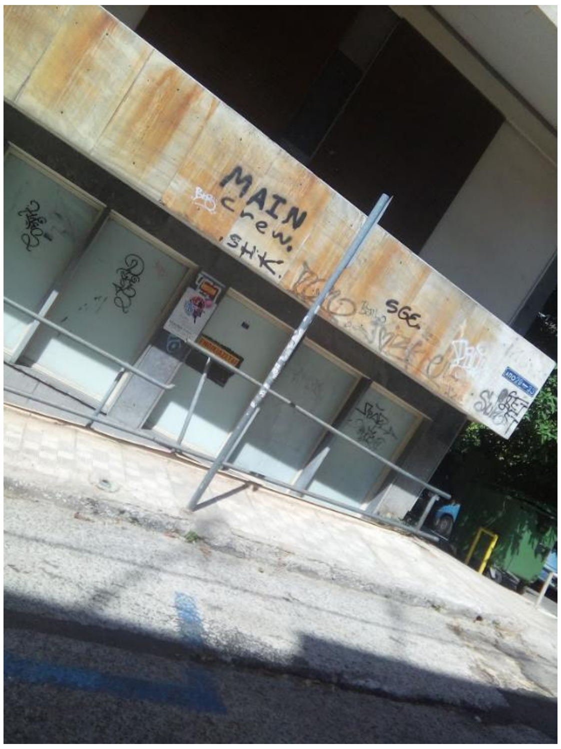


Figure 39: Rotated graffiti tags

Annotating the collected images

After having collected a sufficient amount of images, they need to be annotated. Annotation, here, is the process of drawing rectangle boundaries around objects in each image and declaring what each object is. In this case, the application will only detect graffiti tags and thus, bounding boxes will be drawn only around graffiti tags. Therefore, the annotation process can be simplified by omitting the object type as it will be the same in all instances. To sum it up, each labelled object will be accompanied by 4 numbers translated into pixels to express the position (where the origin of the image is considered the top-left corner) and size of the bounding box in the image:

- the x coordinate of the bounding box
- the y coordinate of the bounding box
- the width of the bounding box
- the height of the bounding box

In this stage, a single file will contain the annotations of all the images in a .json format. This file will be further processed in future steps of the development. Specifically, the file will initially contain 2 keys:

- Images, where all the images are listed with attributes, such as file path and a unique generated id.
- Answers, where an array of annotations (each having x, y, width and height attributes) of each image (using its id) is listed.

In the following example, there is an example of how the annotations of one image are represented. Firstly, the image is declared as an object in the "images" array and it contains a unique id for the image and the location of where the image is saved. Secondly, in the "answers" object the id of the image is used as the key for the nested object so it knows to which image it is referring to. In the id object, there is an array of 3 annotation objects, which means this image contains 3 bounding boxes, each with different positions and sizes.

```
{
    "images": [
        // Previous image
        {
            "_id": "5bafe8650f64191b3a202565",
            "name": "image_153825494209238.jpg ",
            "path": "images/image_153825494209238.jpg",
        }
        // Next image
    ],
    "answers": {
        // Annotations of previous image
        "5bafe8650f64191b3a202565": [
            {
                "x": 1050,
                "y": 1777,
                "width": 808,
                "height": 676
            },
            {
                "x": 255,
                "y": 863,
                "width": 901,
                "height": 561
            },
            {
                "x": 1394,
                "y": 948,
                "width": 820,
                "height": 646
            }
        ]
        // Annotations of next image
    }
}
```

It should be noted that an image with no graffiti tags will be still have its own id object with an empty array of annotation objects.

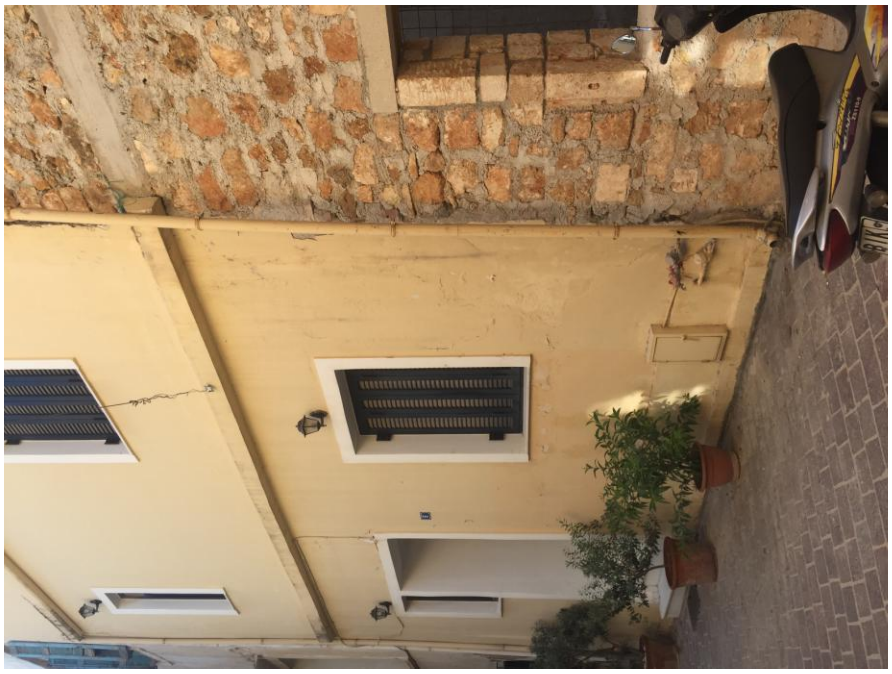


Figure 40: Image with no graffiti tags

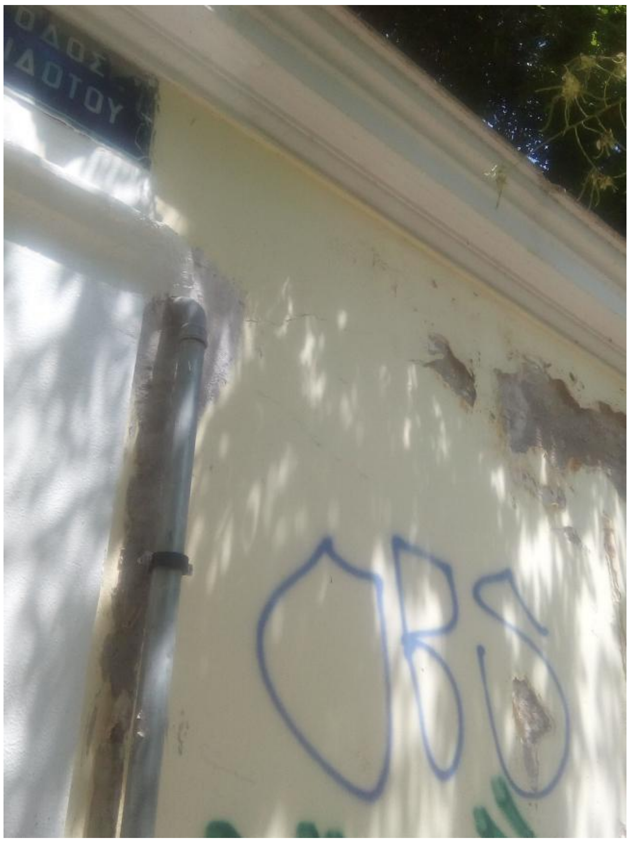


Figure 41: Image with a single graffiti

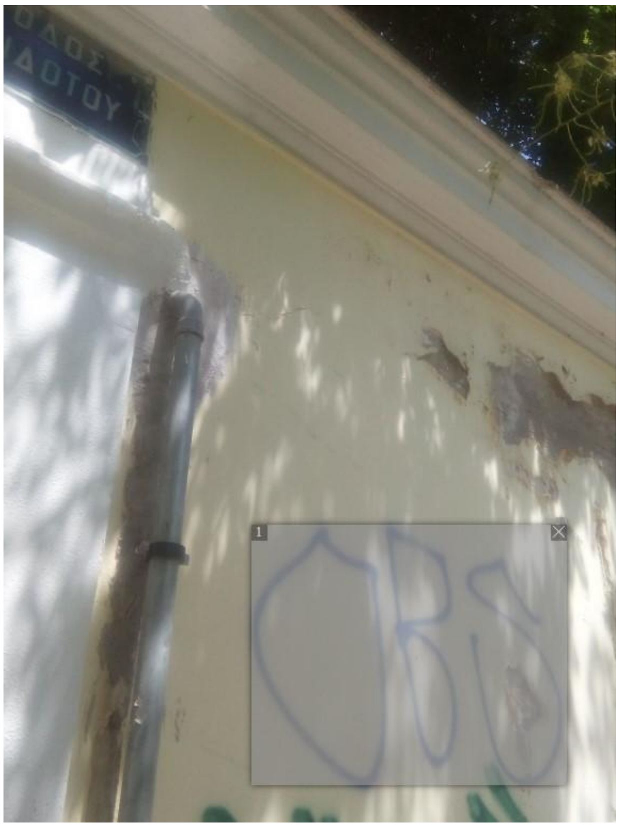


Figure 42: Annotated image with a single bounding box

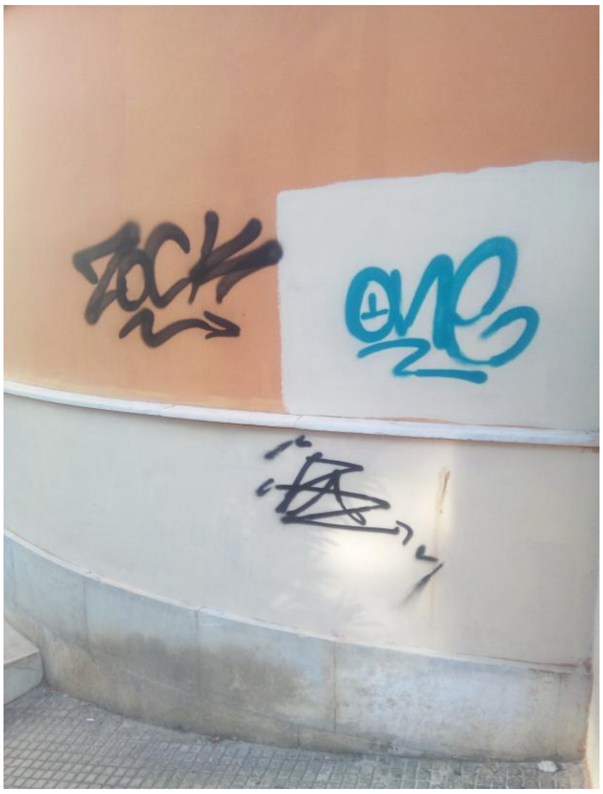


Figure 43: Image with graffiti tags on backgrounds with different colors

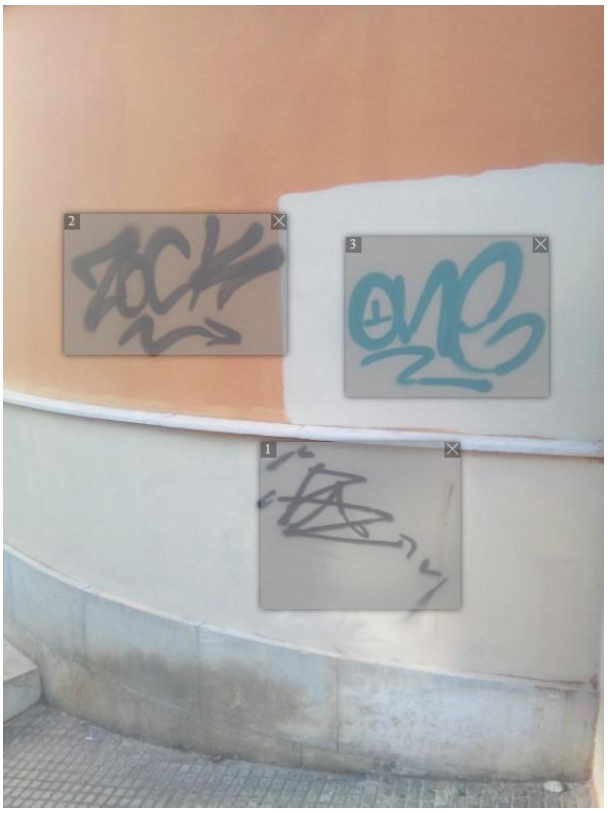


Figure 44: Annotated image with multiple bounding boxes, each for a graffiti tag

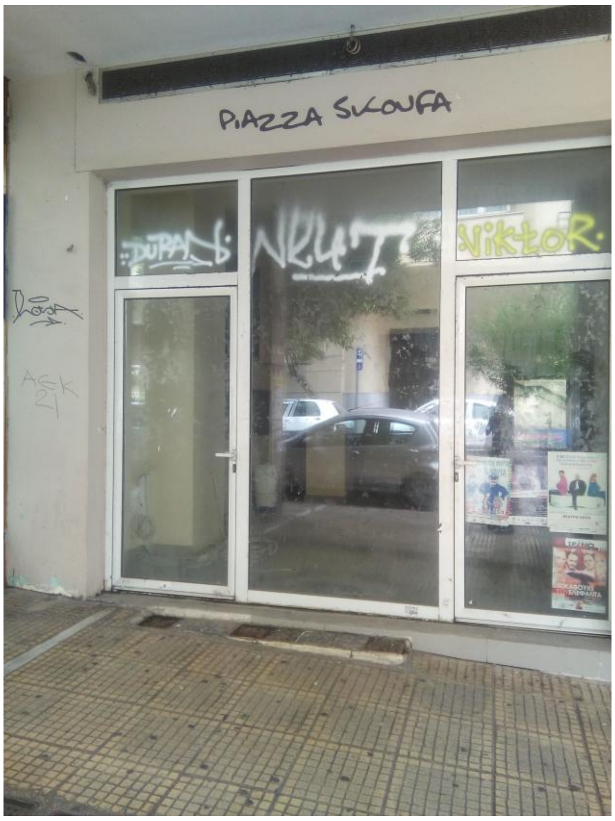


Figure 45: Image with multiple graffiti tags on reflective surface

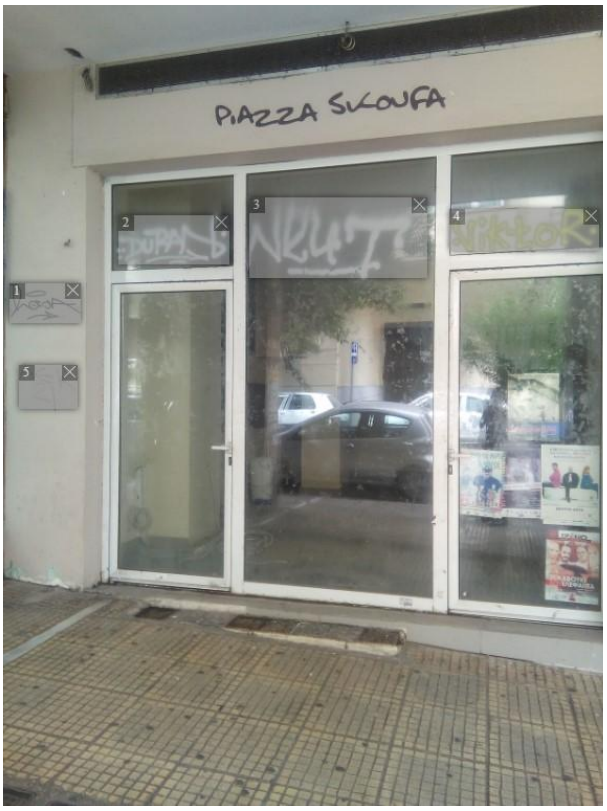


Figure 46: Annotated image with its bounding boxes

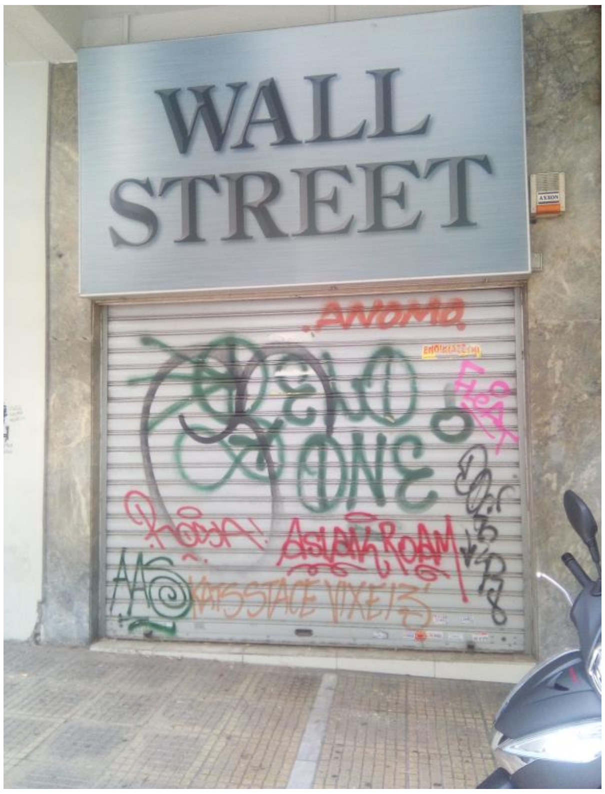


Figure 47: Image with overlapping graffiti

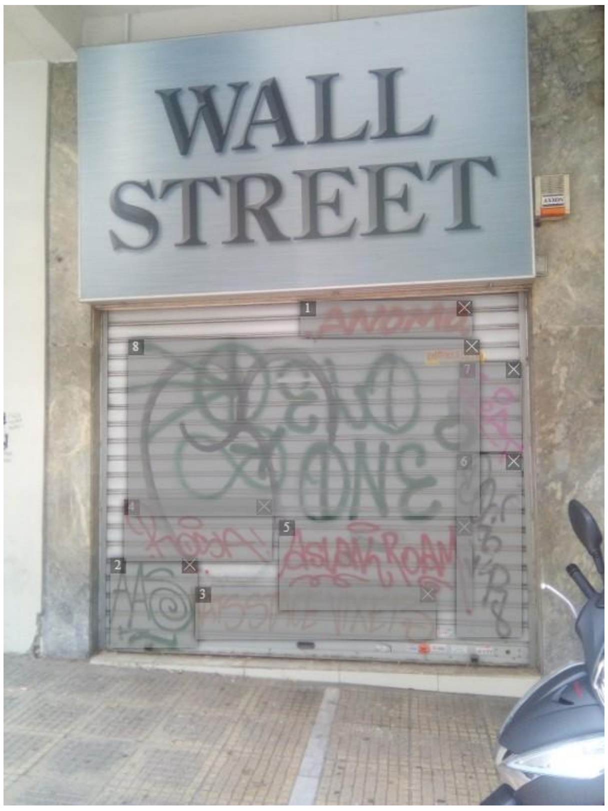


Figure 48: Annotated image with multiple partially-overlapped bounding boxes

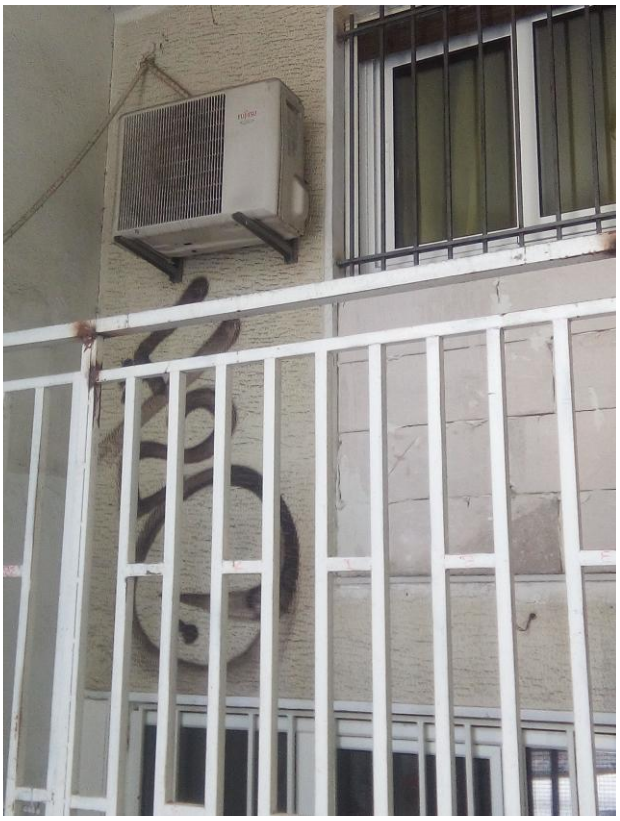


Figure 49: Image with graffiti tag occluded by a fence

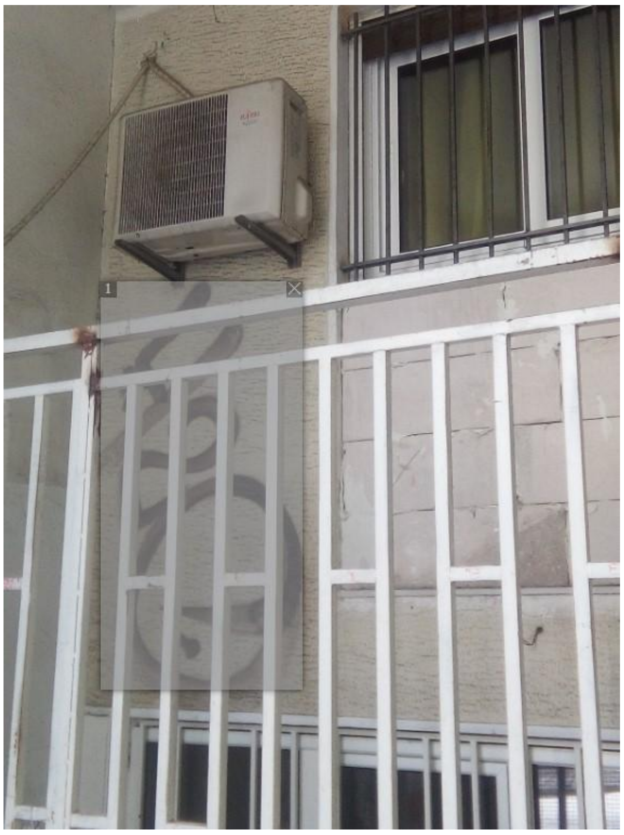


Figure 50: Annotated image with its bounding box

Analyzing and adjusting the images

In this step, an analysis of the image sizes will be done. In the case the images are too large, the images will be downscaled in an attempt to reduce the data for faster training of the model. The following snippets of code will walk through how the image sizes of the dataset are visualized. Firstly, the necessary libraries are imported for reading images and json files and making plots:

```
import os
import json
import numpy as np
from PIL import Image
from matplotlib import pyplot as plt
```

The arrays of the image sizes and the bounding box sizes are initialized, as well as the directories of the needed files:

```
img_widths   = []  # widths of the images
img_heights  = []  # heights of the images
box_widths   = []  # widths of the bounding boxes
box_heights  = []  # heights of the bounding boxes

img_dir   = './images/'
json_file = './annotations.json'
```

Finally, the arrays can be populated by iterating through the files:

```
# Read json
with open(json_file) as json_data:
    d = json.load(json_data)

# Read images
for i in range(len(d['images'])):
    # Record image sizes
    fn =  str(d['images'][i]['name'])
    img = Image.open(os.path.join(img_dir,fn))
    img_widths.append(img.size[0])
    img_heights.append(img.size[1])

    # Get image id
    id = d['images'][i]['_id']

    # Record all bounding box widths and heights
    for j in range(len(d['answers'][id])):
        box_widths.append(d['answers'][id][j]['width'])
        box_heights.append(d['answers'][id][j]['height'])
```

Now that the arrays are populated, the data can be visualized in plots:

```
plt.figure(figsize=(8, 6))
plt.subplot(2, 2, 1)
plt.hist(np.array(img_widths))
plt.title('image widths')
plt.subplot(2, 2, 2)
plt.hist(np.array(img_heights))
plt.title('image heights')
plt.subplot(2, 2, 3)
plt.hist(np.array(box_widths))
plt.title('box widths')
plt.subplot(2, 2, 4)
plt.hist(np.array(box_heights))
plt.title('box heights')
plt.tight_layout()
plt.show()
```

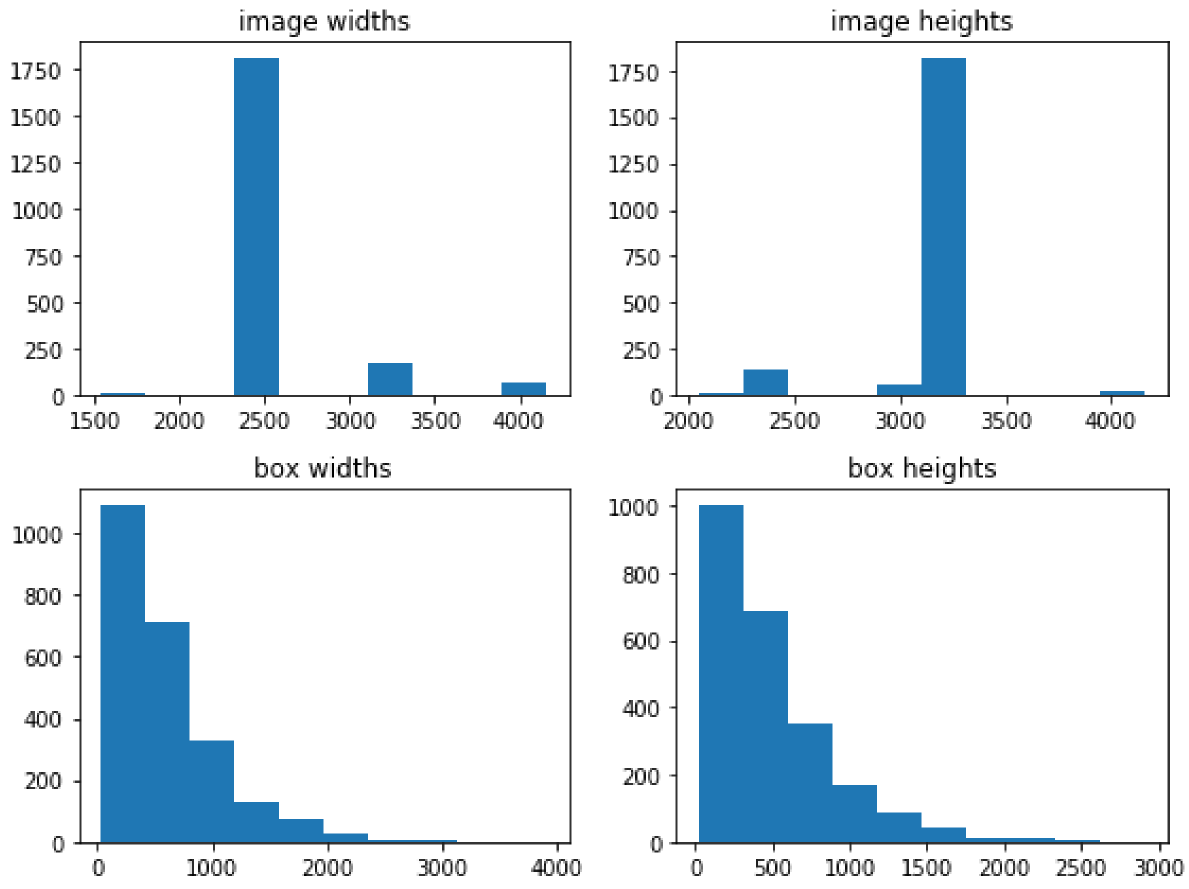


Figure 51: Histograms of image sizes (top) and bounding box sizes (bottom)

As shown, many images are huge, some even reaching 4000 pixels in one (if not both) dimensions, which is way too large. Most images were taken from a single phone with a camera resolution of 2448 x 3264. A large portion of the bounding boxes are reaching the 700-1000 pixels territory, which is large as well. This means that many of the graffiti tags are often occupying around half of the image. In the following figure a scatter plot is shown to visualize in an alternative way the density of the bounding box sizes:

```
plt.figure(figsize=(5, 5))
plt.subplot(1, 1, 1)
plt.scatter(np.array(box_widths),np.array(box_heights))
plt.title('box widths/heights')
plt.xlabel('width')
plt.ylabel('height')
plt.show()
```

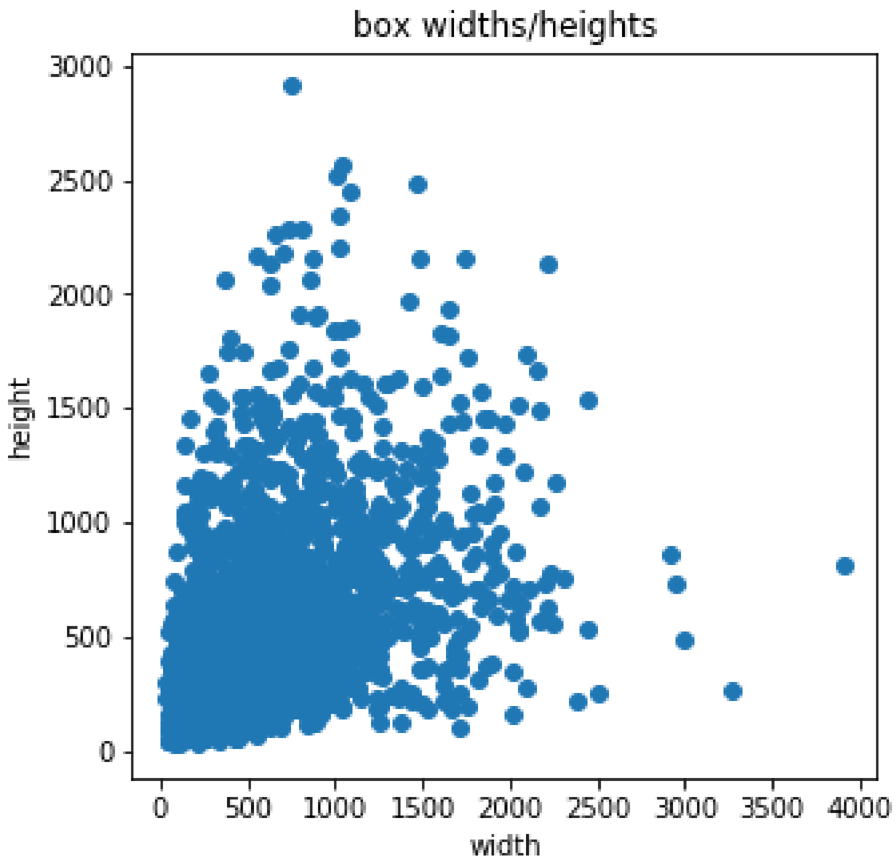


Figure 52: Scatter plot of the bounding box sizes

Most bounding boxes are between (near) 0 and 1500 pixels in any one of the dimensions, while few have a higher amount of pixels, even exceeding the 2500 pixels. If the algorithm received such large images and bounding boxes it would be take a lot of time to reach the desired accuracy as it has to process a lot of pixels. The images will be downscaled in order to reduce that training time. This might conclude to a slightly worse accuracy but the training time reduction is so much more that makes this compromise worth it.

The images will be downscaled to 720x960 pixels as they are numbers that are multiples of 16, which is a requirement of other experimental ML frameworks. The aspect ratio of each image will be kept the same to avoid distortion. Black pixels are filled (if needed) to the sides of each image to maintain the same aspect ratio. Since the images are being downscaled, so do the bounding boxes making sure their position is set accordingly and not left out of bounds. Other frameworks work best with bounding box sizes in the range of 50 and 500 pixels. After the resizing process, the images will be checked with the same histogram and scatter plots to verify the new sizes are within the expected results. Without further ado, the necessary libraries are imported:

```
import os
import json
import numpy as np
from PIL import Image
```

The desired image size and the bounding box arrays are declared:

```
size = [720, 960]  # All input images would be resized accordingly
box_widths  = []   # widths of the bounding boxes after resizing
box_heights = []   # heights of the bounding boxes after resizing
```

The input and the output directories are defined:

```
input_dir  = './images/'
output_dir = './resized_images/'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
json_file  = './annotations.json'
```

Lastly, the resizing of each image is done, and the bounding box sizes are adjusted. Very small bounding boxes (under 20 pixels), which have small significance, are deleted to further reduce data.

```
# Read json
with open(input_dir+json_file) as json_data:
    d = json.load(json_data)

for i in range(len(d['images'])):
    # Resize image
    fn =  str(d['images'][i]['name'])
    im = Image.open(os.path.join(input_dir,fn))

    # Pick the side which needs to be scaled down the most
    scale = min(float(size[0])/im.size[0],float(size[1])/im.size[1])

    im = im.resize((int(im.size[0]*scale),int(im.size[1]*scale)),
                    Image.ANTIALIAS)
    # Padding on either right or bottom if necessary
    im = im.crop((0,0,size[0],size[1]))
    resized_path = output_dir + fn
    # Save the resized image
    im.save(resized_path)

    id = d['images'][i]['_id']
    previous = 0

    # Modify bounding boxes to match image scaling
    for j in range(len(d['answers'][id])):
        x = int(round(d['answers'][id][j-previous]['x'] * scale))
        y = int(round(d['answers'][id][j-previous]['y'] * scale))
        w = int(round(d['answers'][id][j-previous]['width'] * scale))
        h = int(round(d['answers'][id][j-previous]['height'] * scale))

        # Fix to avoid x, y, w, h out of bound
        if x < 0:
            w += x
            if w < 0: w = 20  # this should not happen
            x = 0
        if y < 0:
            h += y
            if h < 0: h = 20  # this should not happen
            y = 0
        if x+w > size[0]:
            w = size[0] - x
        if y+h > size[1]:
            h = size[1] - y

        if (w < 20 and h < 20):
            del d['answers'][id][j-previous]
            previous = previous + 1
        else:
            d['answers'][id][j-previous]['x'] = x
            d['answers'][id][j-previous]['y'] = y
            d['answers'][id][j-previous]['width'] = w
            d['answers'][id][j-previous]['height'] = h
            box_widths.append(w)
            box_heights.append(h)

# Save the updated JSON file
with open(output_dir + 'resized_annotations.json', 'w') as fp:
    json.dump(d, fp, indent=0)
```

With the resized images and annotations, we can now visualize the results:

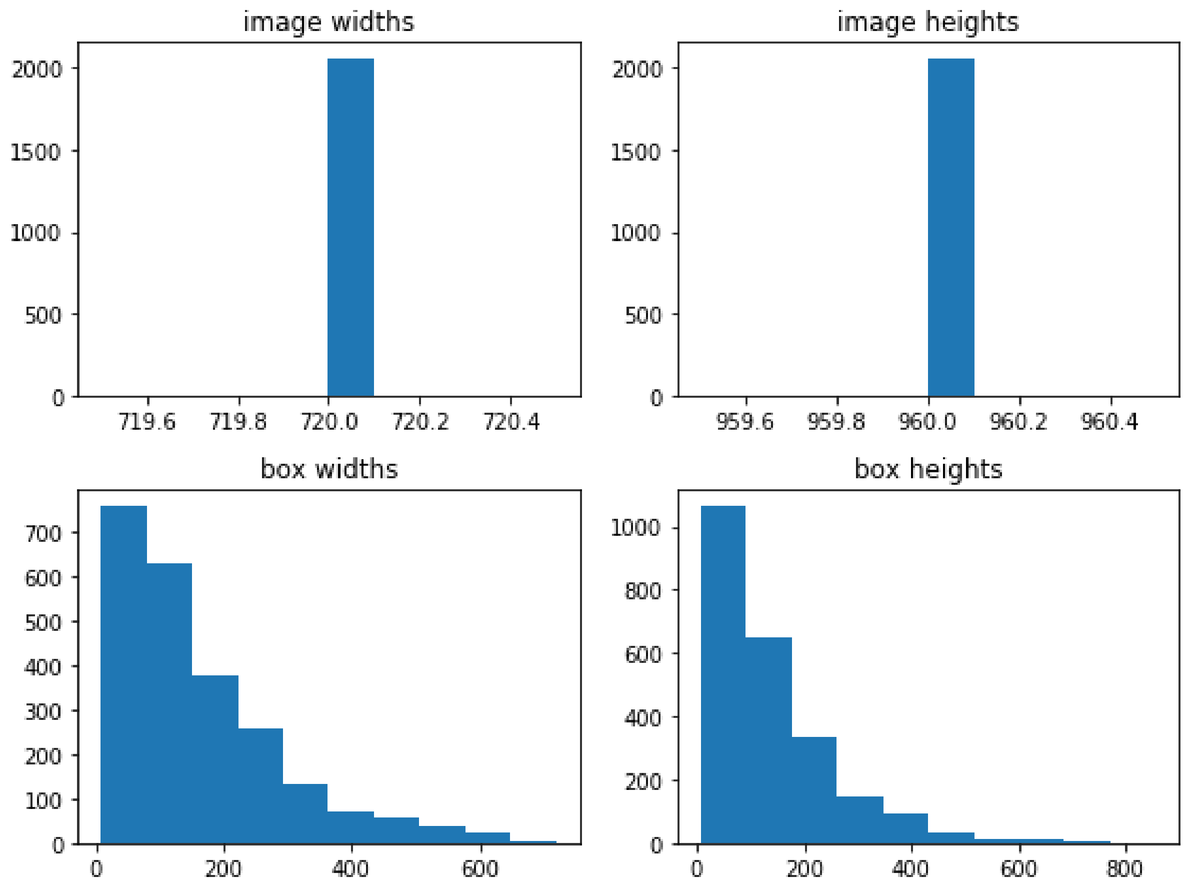


Figure 53: Histograms of resized image sizes (top) and resized bounding box sizes (bottom)

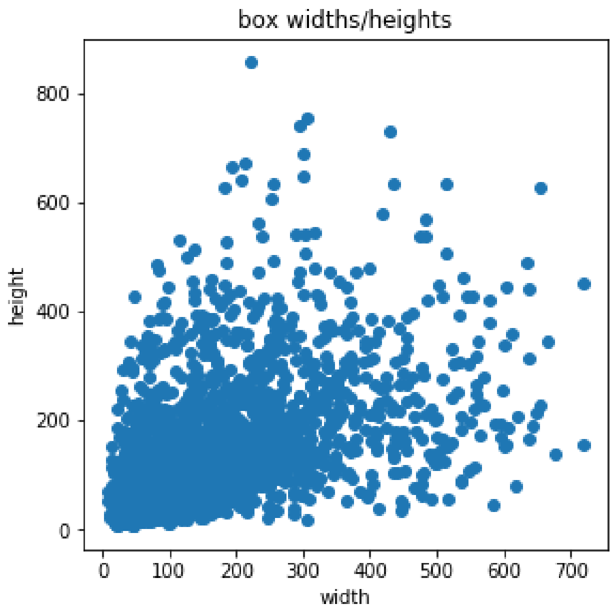


Figure 54: Scatter plot of the resized bounding box sizes

As shown by the plots, all images now are of a single size and the bounding boxes have reduced, as well. Most seem to be in the range of 20 and 500 pixels in one dimension, which is optimal for frameworks to work with. The largest bounding boxes are about 700-800 pixels in each dimension. Do note that many of the images did not need to be padded as the new aspect ratio (720/960 = 0.75) is the same as the original one (2448/3264 = 0.75), which is one of the reasons how the new resolution was decided. Here is a sample image with padding after the resizing:

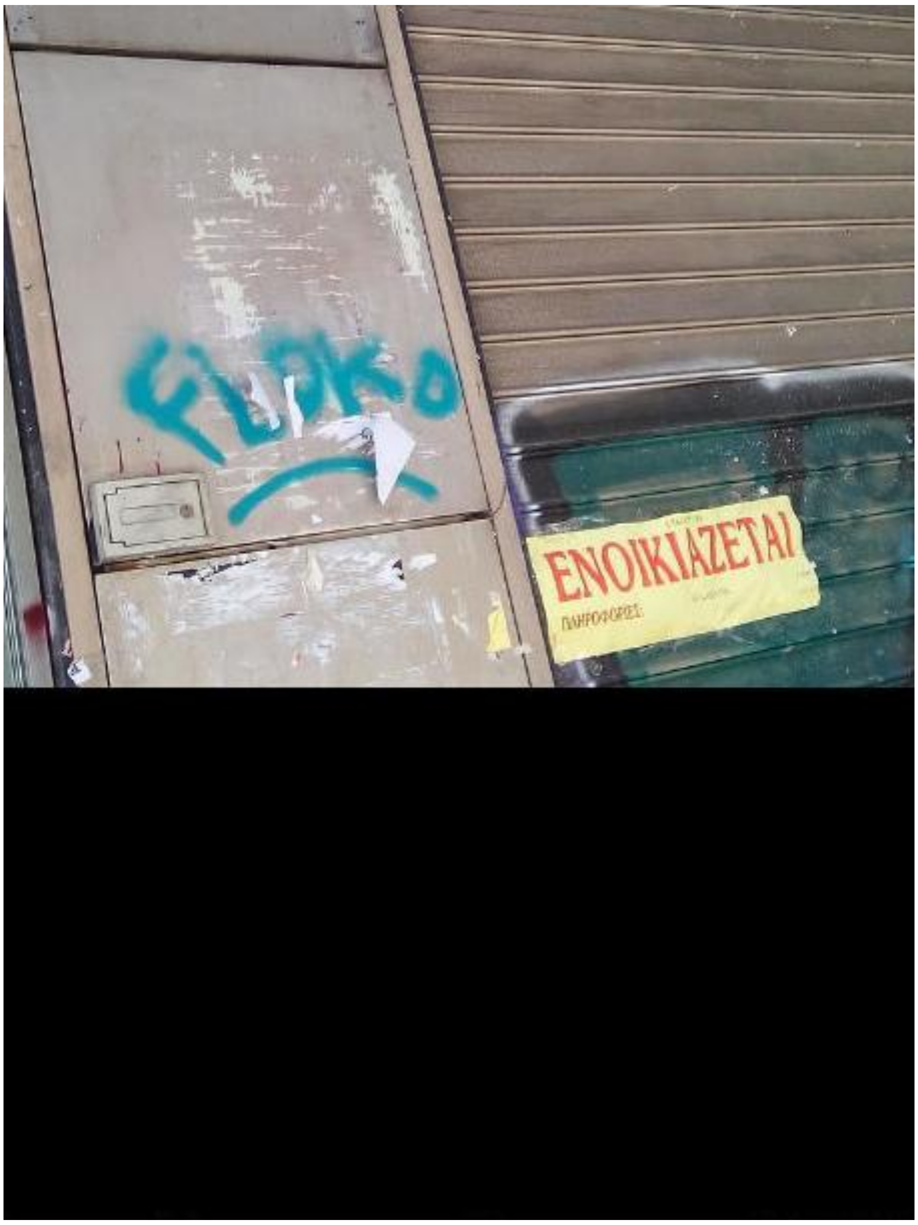


Figure 55: A resized image with padded with black pixels to maintain aspect ratio

Splitting the data into train and test subsets

In this section, the images must be divided to 2 subsets, the train subset and the test one. In a later stage, the model will be training on the images in the train subset, while it will be evaluating its deviation from the desires results based on the images of the test subset. The reason that a different set of images is used to evaluate error is because the model already knows the images of the training set to a degree and thus, it’s biased. It’s like a student prepares for an exam by solving an array of exercises. If the exam contained exactly the same exercises, the knowledge of the student couldn’t be properly assessed to understand whether he/she has truly grasped the concept. Usually, most data are in the training set and a small portion is reserved for the test set. That’s because the model needs as many data as it can to achieve high accuracy. Here, the training subset will contain 80% of the total images and 20% will be the test subset, though this split ratio can be fine-tuned to potentially achieve better results at the end. With the help of a python script the separation of the training data from the test data is achieved. The following script takes as input the json file that was created previously along with the images it is referencing to and creates 2 folders (one containing the training images and the other containing the test images) along with 2 csv files (one containing all the bounding boxes of the training images and the other containing all the bounding boxes of the test images).

```
import os                    # for creating folders
import csv                   # for creating and writing to a csv file
import json                  # for reading a json file
from PIL import Image        # for reading img files
from shutil import copyfile # for moving files to other directories
import random                # for generating a random number

trainRatio = .8 # define the percentage of training images

# create the split folders
if not os.path.exists("./train/"):
    os.makedirs("./train/")
if not os.path.exists("./test/"):
    os.makedirs("./test/")

# open json file and create 2 csv files (one for train data, one for test
data)
with open("resized_annotations.json") as json_data, \
open("train_annotations.csv","wb+") as train_data, \
open("test_annotations.csv","wb+") as test_data:
    d = json.load(json_data)
```

```
# define the attributes of each image that is essential
columnName = ["filename","width","height", \
"class","xmin","ymin","xmax","ymax"]

# create a writer for each file to use later
train = csv.writer(train_data)
test = csv.writer(test_data)

# set the attributes names as the 1st row for each csv file
train.writerow(columnName)
test.writerow(columnName)

# for each image
for i in range(len(d['images'])):
    # store the image name
    fn = str(d['images'][i]['name'])

    # Copy the image and select csv write according to trainRatio
    if random.random() < trainRatio:
        csvOutput = train
        copyfile("./images/"+fn,"./train/"+fn)
    else:
        csvOutput = test
        copyfile("./images/"+fn,"./test/"+fn)

    # read image
    im = Image.open('./images/'+fn)

    # Acquire image id
    id = d['images'][i]['_id']

    # for each bounding box, fill each column with appropriate info
    for j in range(len(d['answers'][id])):
        annotation = d['answers'][id][j]
        csvOutput.writerow([fn,
                            im.size[0],
                            im.size[1],
                            "Graffiti",
                            annotation["x"],
                            annotation["y"],
                            annotation["x"]+annotation["width"],
                            annotation["y"]+annotation["height"]])
```

Generating tensorflow records

In this section, the process of creating TFRecord files, which are needed to train the object detection model, is being covered. Tensorflow's object detection API can receive the annotations of the images in the TFRecord file format, which is tensorflow's own binary storage format. Having the data in a binary format improves the performance of the training, especially when using large datasets. Luckily, the conversion of the csv files to TFRecords has already been implemented by someone else and has been uploaded as a python script [58]. A slight modification in the script is necessary to tailor it to the needs of this application:

```
def class_text_to_int(row_label):
    if row_label == 'Graffiti': # changed label to Graffiti
        return 1
    else:
        None
```

Using the script, a train.record file is generated from the annotations of the training images and a test.record file is generated from the annotations of the test images.

Getting a pre-trained model and a configuration file

At this stage, the training algorithm has all the data it needs to start training. However, if the training starts now it will be a learning process that starts from scratch. Therefore, transfer learning will be utilized to improve training time. For this approach, a pre-trained model is required, which is accompanied by a configuration file. The configuration file is a group of settings that can fine-tune the training of the algorithm. Luckily, the object detection API already offers a plethora of models [59] and configurations [60] on GitHub.

Since this application will run on a mobile device, the inference of the model must be as fast as possible, while maintaining the best accuracy possible. It must also be taken into account that each pre-trained model have had different datasets they learned from. There are many available datasets, such as:

- COCO, which contains images with many different common objects
- Kitti, which contains images usually with vehicles and pedestrians and thus, is optimal for autonomous driving applications
- Open Images, which contains also many different categories, such as COCO
- iNaturalist Species, which contains images of many animal species
- AVA, which contains many video clips of labeled human activities

In this application, the "ssd_mobilenet_v1_coco" model will be used, which provides an average inference time of 30ms using a 600x600 image with an accuracy of 21%. This model is accompanied with the configuration file “ssd_mobilenet_v1_coco.config” that was used in the pre-training of the model. The configuration file requires some changes before it’s used. The key points are:

- From

  ```
  num_classes: 90
  ```

  to

  ```
  num_classes: 1
  ```

- From

  ```
  learning_rate: {
    exponential_decay_learning_rate {
      initial_learning_rate: 0.004
      decay_steps: 800720
      decay_factor: 0.95
    }
  }
  ```

  to

  ```
  learning_rate: {
    exponential_decay_learning_rate {
      initial_learning_rate: 0.004
      decay_steps: 2000
      decay_factor: 0.95
    }
  }
  ```

- From

  ```
  fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED/model.ckpt"
  ```

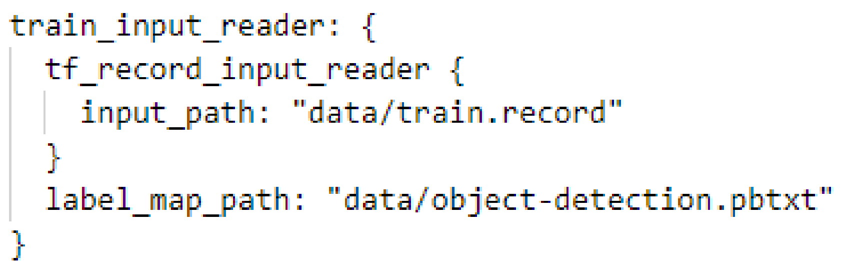
  to

  ```
  fine_tune_checkpoint: "ssd_mobilenet_v1_coco_2018_01_28/model.ckpt"
  ```

- From

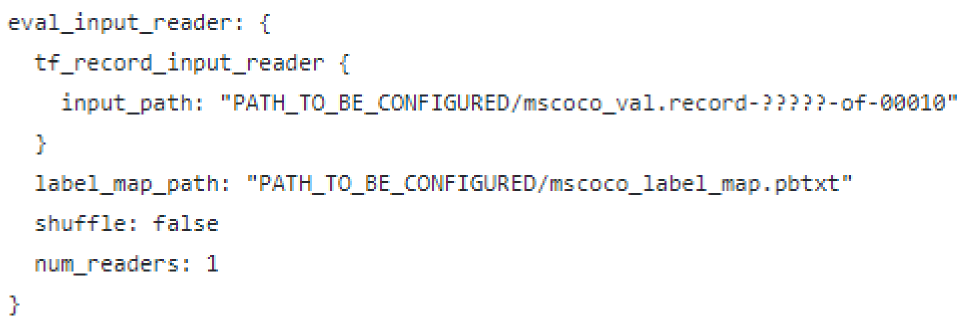
  ```
  train_input_reader: {
    tf_record_input_reader {
      input_path: "PATH_TO_BE_CONFIGURED/mscoco_train.record-?????-of-00100"
    }
    label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt"
  }
  ```

  to



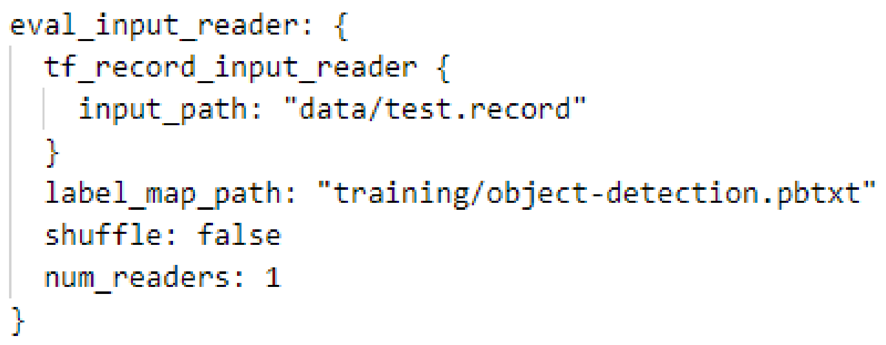
```
train_input_reader: {
  tf_record_input_reader {
    input_path: "data/train.record"
  }
  label_map_path: "data/object-detection.pbtxt"
}
```

- From



```
eval_input_reader: {
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED/mscoco_val.record-?????-of-00010"
  }
  label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

to



```
eval_input_reader: {
  tf_record_input_reader {
    input_path: "data/test.record"
  }
  label_map_path: "training/object-detection.pbtxt"
  shuffle: false
  num_readers: 1
}
```

It should be noted that the learning rate can be set more “aggressive” in the beginning of the training (larger initial learning rate such as 0.01), while during and at the end it can be set to lower values to avoid overshooting. Also, the batch_size can be decreased in case the GPU doesn’t have enough VRAM.

Lastly, the label map needs to be created. Since the id = 0 is reserved, the “Graffiti” tag will be with id = 1 and will look like this in a new “object-detection.pbtxt” file:

```
item {
  id: 1
  name: 'Graffiti'
}
```

To summarize where each file should be, in the Model > research > object_detection directory (where the tensorflow object detection API was installed) the hierarchy of the relevant files should be:

Object_detection

- data/
  - train.record
  - test.record
  - object-detection.pbtxt
- images/
  - train/
    - all the training images
  - test/
    - all the testing images
- training/
  - object-detection.pbtxt
  - ssd_mobilenet_v1_coco.config
- ssd_mobilenet_v1_coco_2018_01_28
  - model.ckpt

Training the model

Finally, the training can begin. Being inside the Object_detection folder the following command can be executed to initiate the process:

```
python legacy/train.py \
        --logtostderr \
        --train_dir=training/ \
        --pipeline_config_path=training/ssd_mobilenet_v1_coco.config
```

After some initialization messages, the command prompt should show the step, the loss in that step, as well as how long it took to complete the last step. The loss should start from a high value and with each step, decrease. There's a point that this loss will reach a very small number and the amount it decreases after that is near 0 with the current input data. At that stage, the developer can fine-tune the configuration file (such as decreasing the learning rate) to attempt reduce the loss even more.

Tensorflow provides a tool called Tensorboard, which makes it easy to monitor the progress of the training. This tool was used to try to refine the values of the configuration file in order such to improve the efficiency of the learning process. The algorithm was trained for a bit over 130.000 decay steps, which for the given GPU was about a week. During this time the training was interrupted multiple times (after having saved the progress to a checkpoint file) to adjust the learning rate, which can be shown in the following graph:

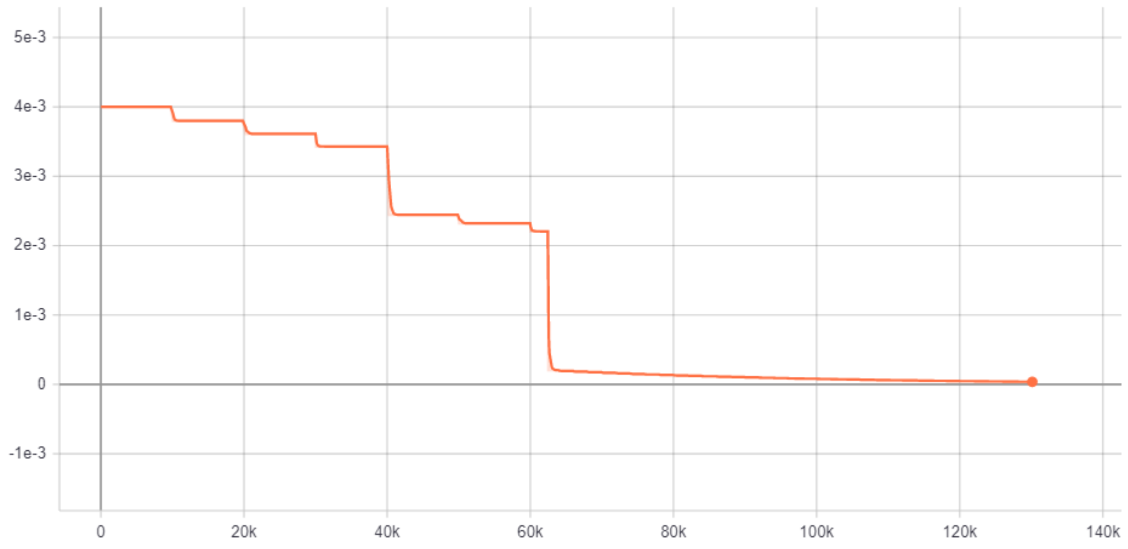


Figure 56: Learning rate over decay steps

As shown by the above graph, the learning rate starts with a value of 0.004 defined by the configuration file. Till it's set again after a checkpoint, the learning rate decreases very slowly by itself, which is barely visible except in the last set (just after step 60.000).

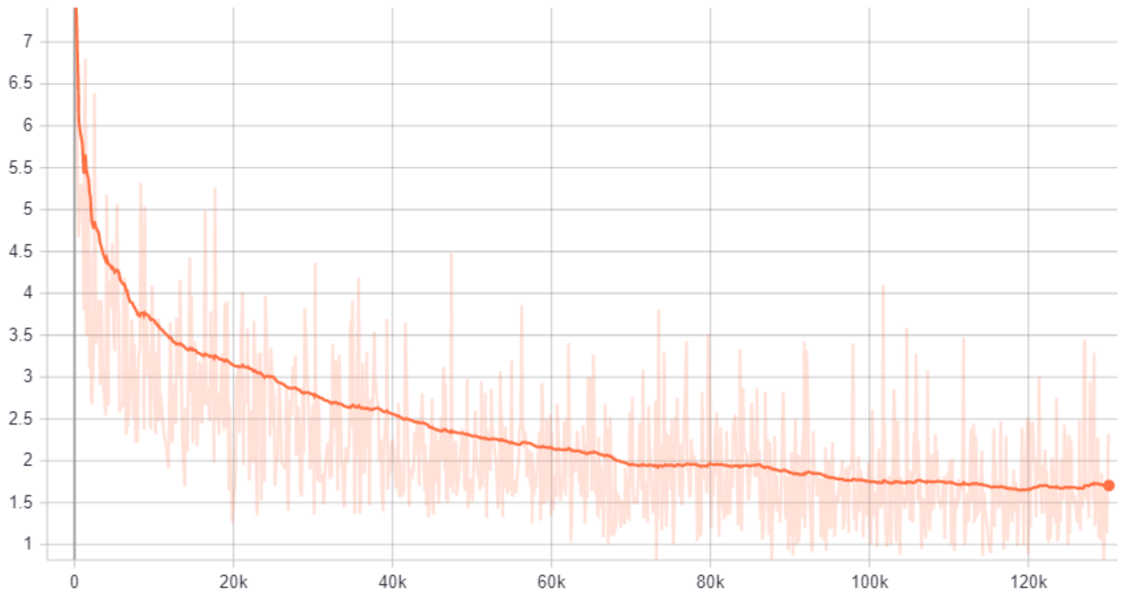


Figure 57: Total loss over decay steps

As expected, the algorithm manages to decrease significantly in the first few steps. After step 90.000 the decrease is barely noticeable and concludes to a loss of around 1.7 at step 130.000.

Evaluating the trained model

After the training has been complete, evaluation of the model is needed. Through the evaluation it is shown what the accuracy of the model is. It also shows how often the model detects an object it isn't supposed to (also known as false positive) or how often it doesn't detect an object it is supposed to (also known as false negative). The following command will produce the mentioned metrics (and more) and store them in the "eval" folder:

```
python legacy/eval.py \
        --logtostderr \
        --pipeline_config_path=training/ssd_mobilenet_v1_coco.config \
        --checkpoint_dir=training/ \
        --eval_dir=eval/
```

The tensorboard tool is used to read and visualize the graphs:

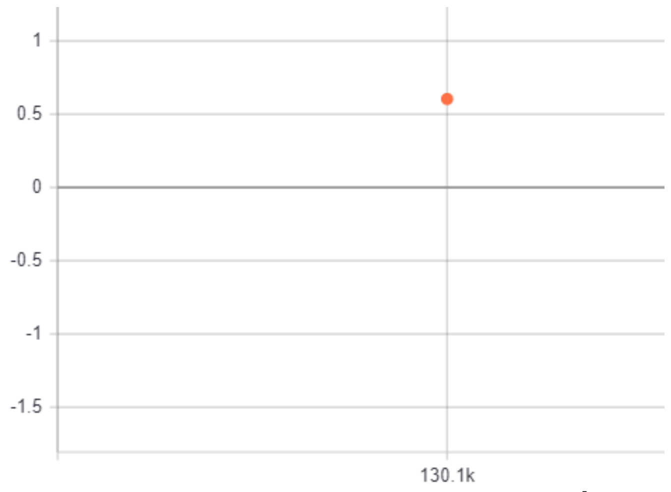


Figure 58: 0.6 mAP@.5 IoU at the 130.000th step

The mean average precision of the trained model is at ~60% with 50% Intersection over Union (IoU). IoU describes how much 2 bounding boxes overlap and it is used to compare how much the predicted bounding box overlaps the correct bounding box (from the annotation process). Often, when 2 bounding boxes overlap by over 50% (which is an adjustable threshold), then it's considered a true positive. From the following graph, it's shown that the accuracy falls the smaller the bounding box (compared to the image size) is, which is quite expected as there are fewer pixels in those bounding boxes to process. This might have increased if we used resized to a larger resolution (than 720x960) but it would also mean that the inference time would increase.

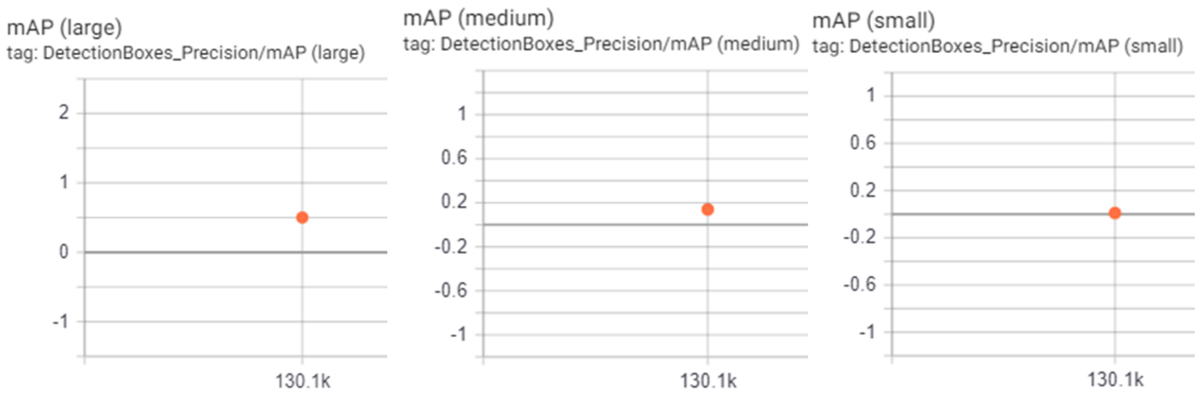


Figure 59: Precision at 50% in large detections, 14% in medium and 1% in small

Another metric is the recall. Just like the precision, recall decreases by a lot in the small-sized detections.

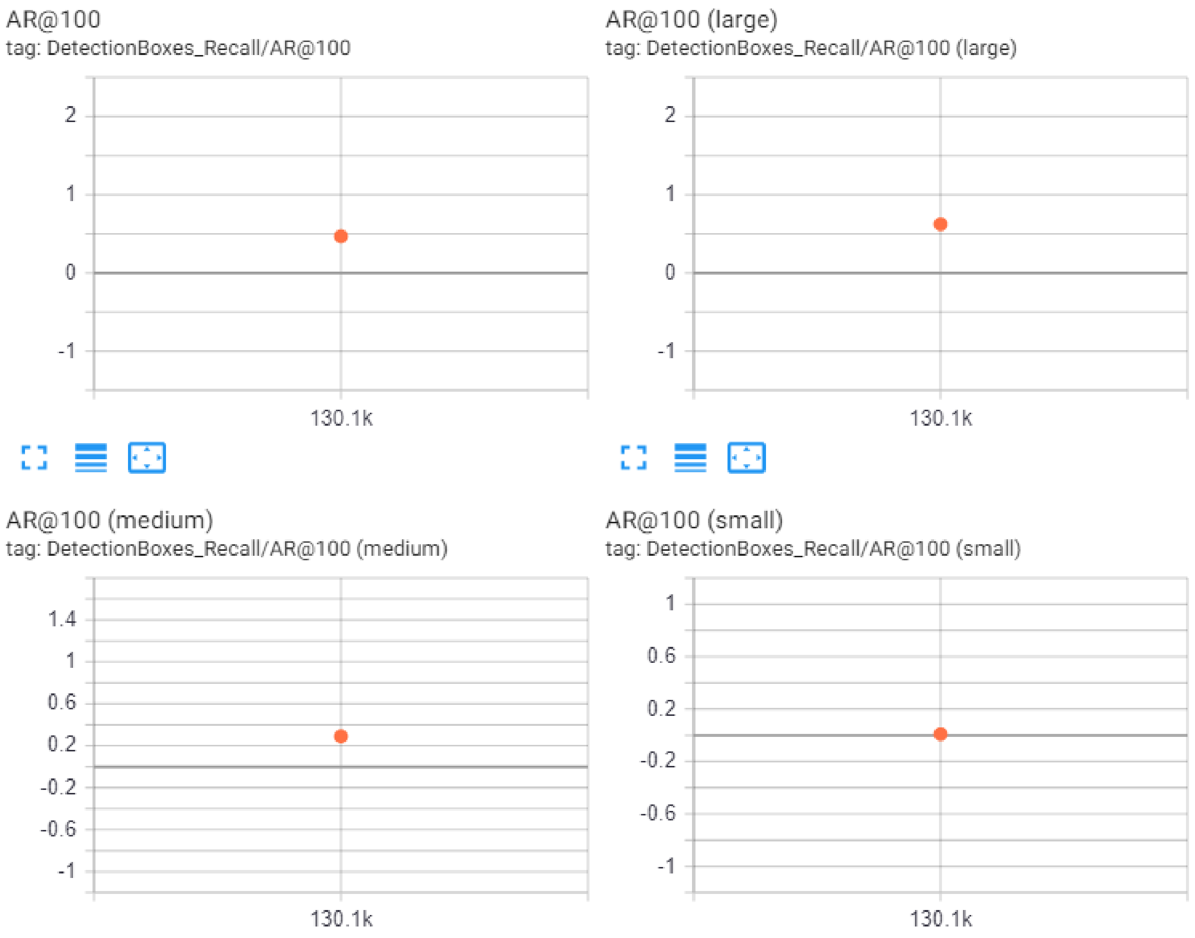


Figure 60: Average Recall with 100 detections at 47% (top-left), 62% with 100 large detections (top-right), 29% with 100 medium detections (bottom-left) and 0.9% with 100 small detections (bottom-right)

Precision and Recall are two of the most important in evaluating the model. Recall means how many of the true positives were found and is calculated as:

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Precision means how many of the returned hits were true positives and is calculated as:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

While it's desired for one metric to be as close to 1 as possible, it also means that the other metric will reduce close to 0, which isn't beneficial for the model. Thus, both need be taken into account to balance them in this Precision-Recall trade-off. There is, also, another metric, which is derived from these two, called F1 score. It ranges from 0 (bad model) to 1 (excellent model) and is calculated as:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Using the mAP@.5IoU (0.6) and the AR@100 (0.47), the result of the F1 score for the graffiti detection model is around 0.53. There is much room for improvement.

Here are some test images after going through the inference process using Tensorboard:

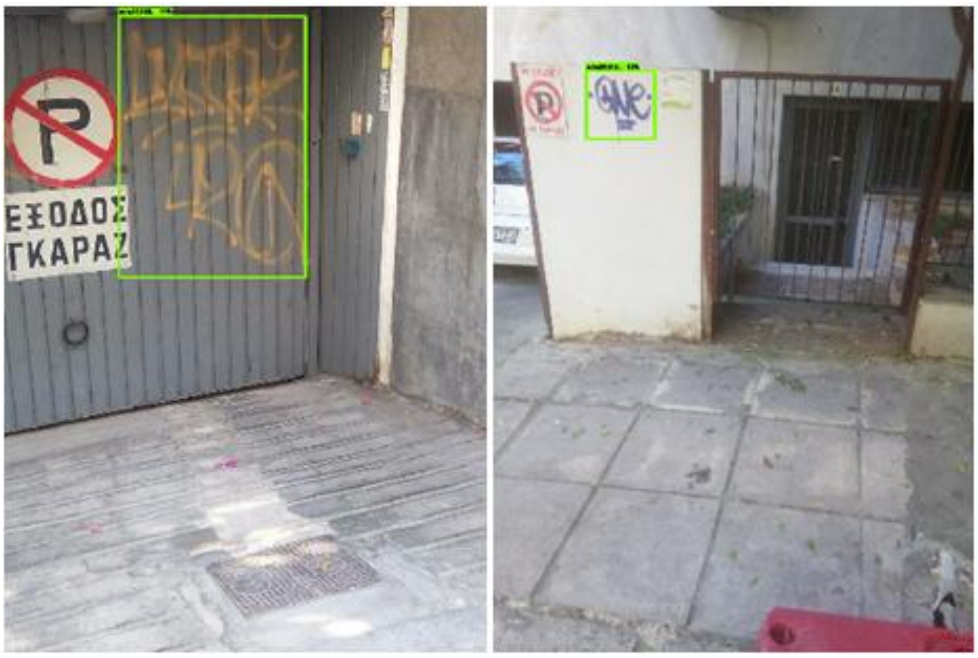


Figure 61: The model is able to confidently distinguish graffiti tags from traffic signs

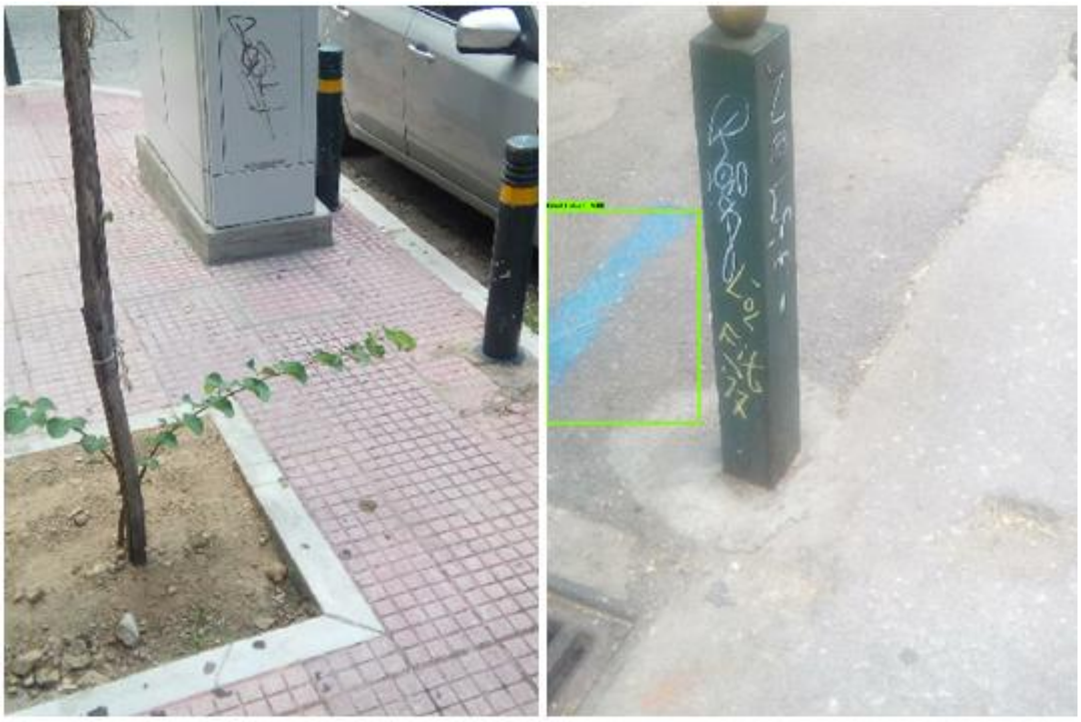


Figure 62: Some detection misses by the model

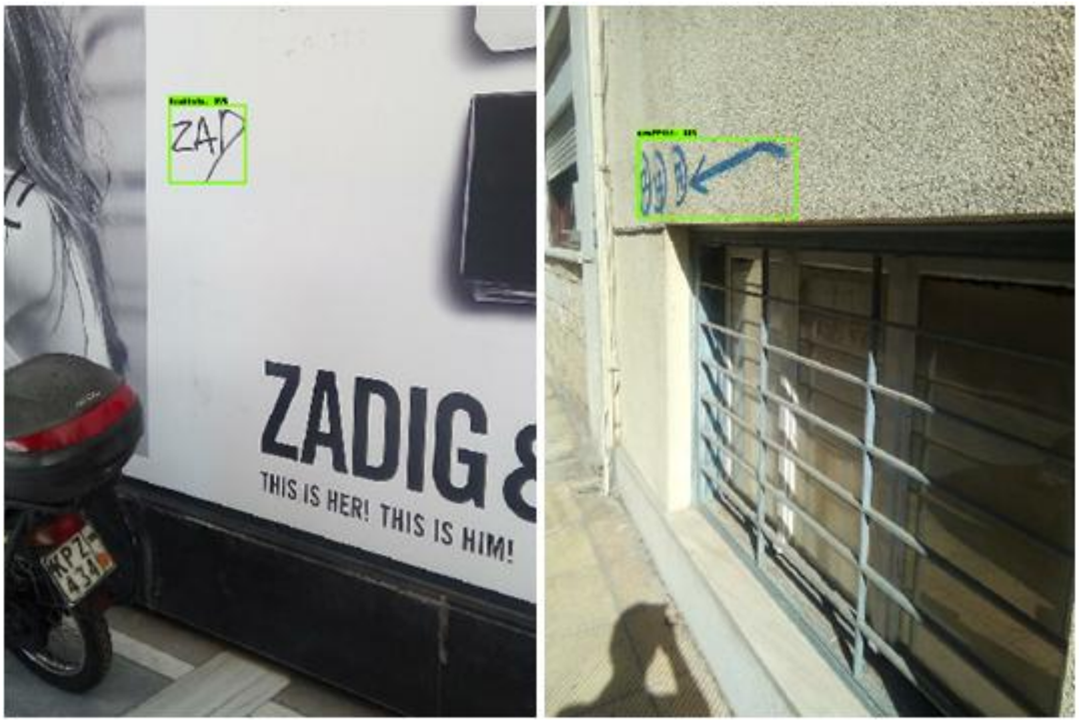


Figure 63: The model can distinguish the handwritten tag from the printed (left), as well as the tag that that is skewed (right)

Exporting the inference graph

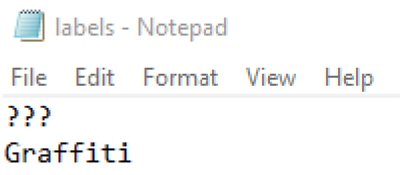
Now that the model is trained, the inference graph is exported. The object detection API already offers a script that produces that file through the command:

```
python export_inference_graph.py \
        --input_type image_tensor \
        --pipeline_config_path training/ssd_mobilenet_v1_coco.config \
        --trained_checkpoint_prefix training/model.ckpt-130060 \
        --output_directory graffiti_inference_graph
```

Where the trained checkpoint prefix should be the last checkpoint (with the largest step number) that has created 3 files:

- model.ckpt-x.data-00000-of-00001
- model.ckpt-x.index
- model.ckpt-x.meta

The above command will save the inference graph called “frozen_inference_graph.pb” in the defined “graffiti_inference_graph” folder inside the object_detection directory. Also, a new text file, called “labels.txt”, need to be created to contain the classes of the objects the application will be detecting. Here, there is only one class (besides the reserved one). The file will look like this:



```
???
Graffiti
```

The inference graph file and the labels map file are the only files needed to start using object detection in any application.

Integrating the frozen graph into the android application

This is the final stage of the project, where the trained model will be put in use in a real-time graffiti detection application. Tensorflow already provides an android camera demo [55], which will be used after some modifications to integrate the model of this project. The demo application contains 4 different sub-applications, of which only the “TF Detect” is relevant. In the README file it’s explained what tools can be used to build the application. Here, Android Studio is used together with Bazel using a linux OS (Ubuntu 16.04).

The first thing to do is to move the necessary files to the appropriate location in the camera demo project. To be more specific, the frozen graph file and the labels map file should be moved inside the assets folder. Secondly, the modifications of specific java file must take place to reference those two files when the application is in use. Specifically, in the src > org > tensorflow > demo directory, there is a file called “DetectorActivity.java”. Upon opening the file, in the first few lines some parameters are defined, including the input model and labels file. The following variables should be set like this:

```
private static final String TF_OD_API_MODEL_FILE =
"file:///android_asset/frozen_inference_graph.pb";
private static final String TF_OD_API_LABELS_FILE =
"file:///android_asset/labels.txt";
```

One can also make adjustments on the MINIMUM_CONFIDENCE_TF_OD_API variable, which determines how confident the application must be to track a detected object. The default value is 0.6f (60%). If it’s required to be more selective in the detections, then the value should increase.

Aside that from above changes, some additional changes were made to support a saving feature. These changes have no relation to the machine learning algorithm itself and are optional. The goal is to save the image with its bounding boxes in order to have the flexibility to further process detected images in the future. The proposed changes will not save a screenshot of the display, but will save the actual image that is fed into the model, which is after the cropping/down-sampling. Without further ado, in the file “DetectorActivity.java” a new variable is introduced together with the other declarations in the beginning:

```
public static boolean SAVE_RESULT_BITMAP = false;
```

This flag determines whether the image currently being processed should be saved or not after the processing is done. Towards the end of the same file, the following code exists:

```
for (final Classifier.Recognition result : results) {
  final RectF location = result.getLocation();
  if (location != null && result.getConfidence() >= minimumConfidence) {
    canvas.drawRect(location, paint);
    cropToFrameTransform.mapRect(location);
    result.setLocation(location);
    mappedRecognitions.add(result);
  }
}
```

After which, the actual saving code should get added, which looks like this:

```
if (SAVE_RESULT_BITMAP) {
  ImageUtils.saveBitmap(cropCopyBitmap);
  SAVE_RESULT_BITMAP = false;
}
```

In the file “CameraActivity.java”, which is in the same folder, the volume-down button is remapped to the newly added saving option, while the volume up remains for activating debug mode. To implement that, the onKeyDown() callback method should be re-written as:

```
@Override
public boolean onKeyDown(final int keyCode, final KeyEvent event) {
  if (keyCode == KeyEvent.KEYCODE_VOLUME_UP
      || keyCode == KeyEvent.KEYCODE_BUTTON_L1
      || keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
    debug = !debug;
    requestRender();
    onSetDebug(debug);
    return true;
  }else if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN){
    Toast.makeText(this, "Image saved!", Toast.LENGTH_SHORT).show();
    DetectorActivity.SAVE_RESULT_BITMAP = true;
    return true;
  }
  return super.onKeyDown(keyCode, event);
}
```

With these adjustments the configuration of the application is now done, which leaves the build and install process, which is the same as any other android application.

Prototype and Use

With the installation of the android application using the .apk file created from the Android Studio’s build, the prototype is created. The application is quite simple to use as it works pretty much on its own once launched. Firstly, the application relevant to this project is called “TF Detect” and carries the tensorflow logo:

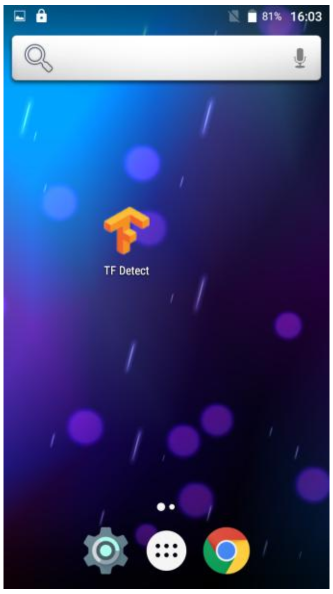


Figure 64: TF Detect app on home screen

Launching the application for the first time requires the grant of some permissions from the user in order for the application to run:

- Access to the camera
- Access to file storage to save images

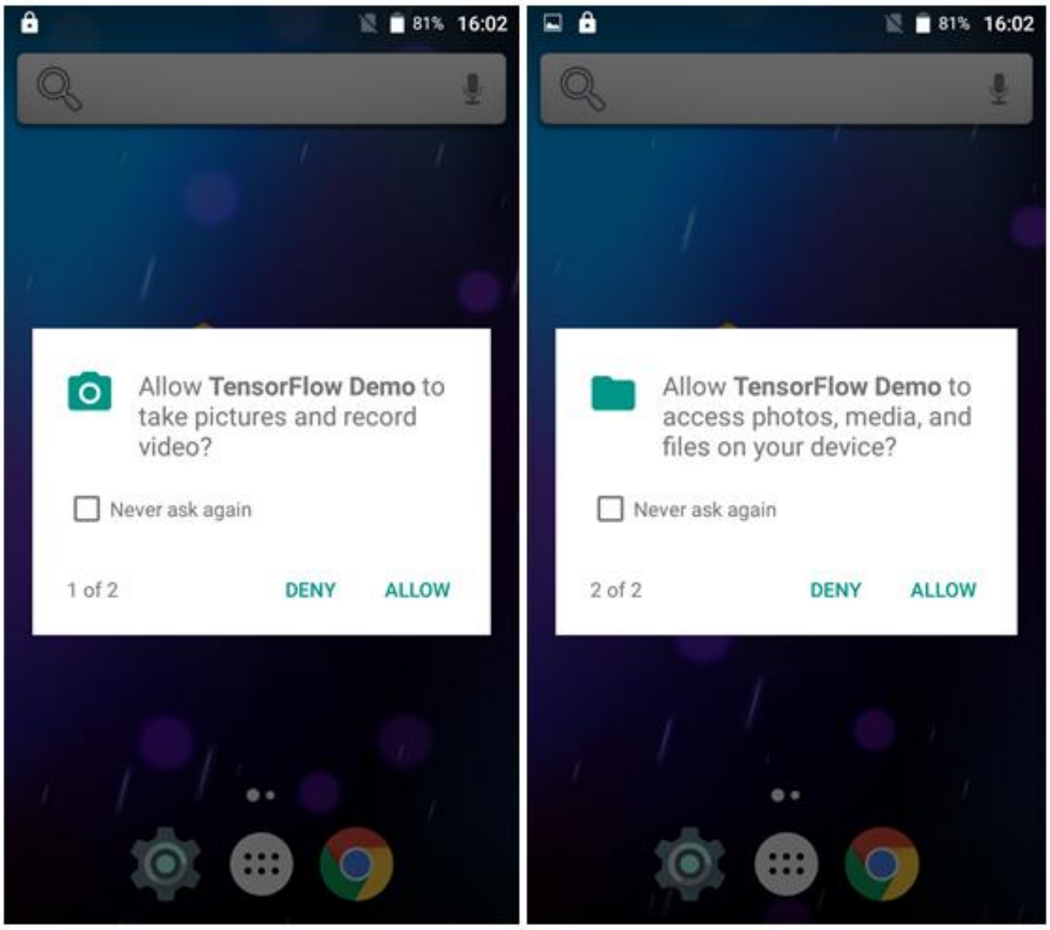
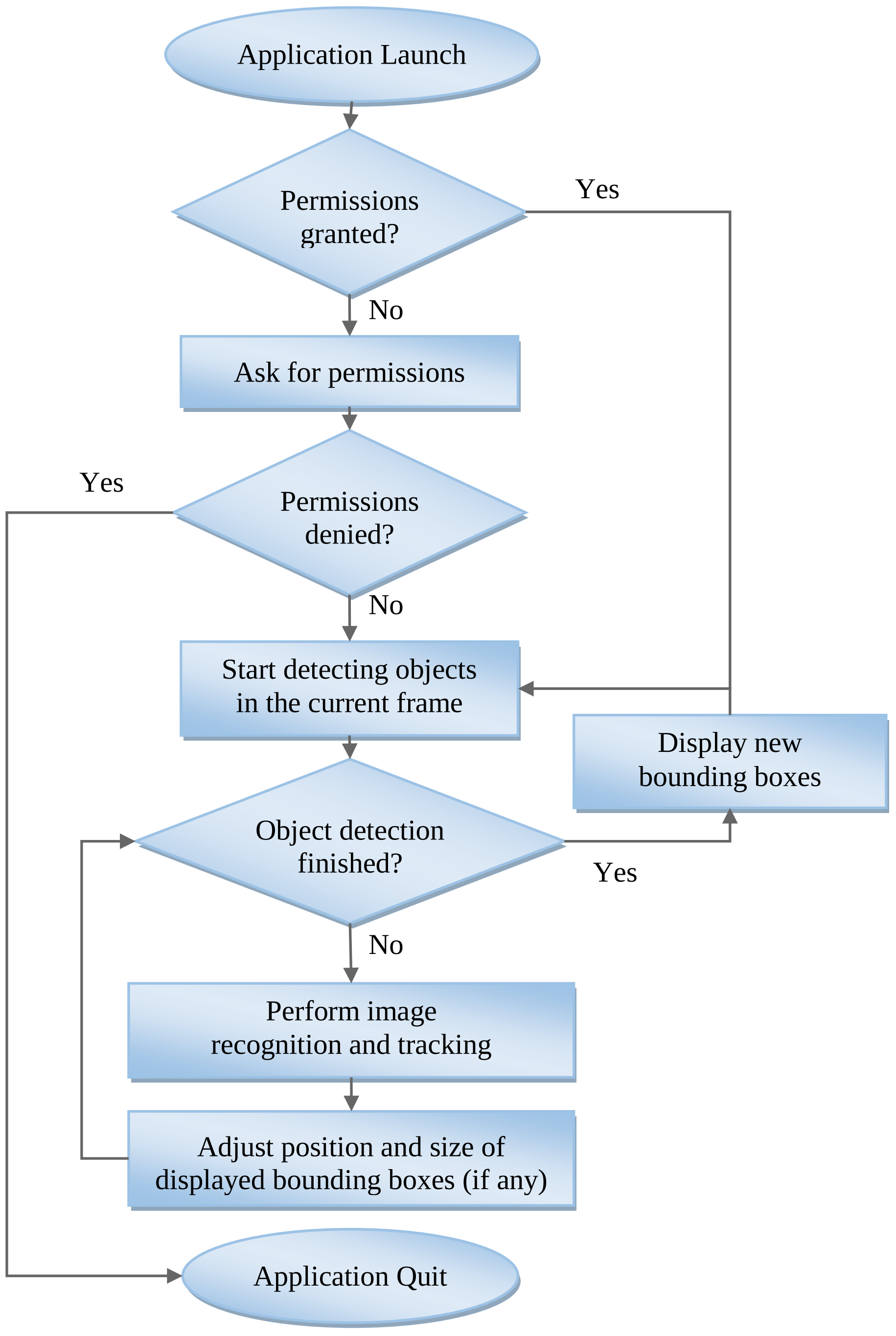

81% 16:02
Allow TensorFlow Demo to take pictures and record video?
Never ask again
1 of 2
DENY
ALLOW
81% 16:02
Allow TensorFlow Demo to access photos, media, and files on your device?
Never ask again
2 of 2
DENY
ALLOW

Figure 65: Permissions required to run the android application

By pressing the volume-down button, the application saves on the device the image that is currently being processed. The application also has a debug mode, activated when pressing the volume-up physical button. This mode shows a lot of information regarding the inference of the current image. One of the stats it shows is the inference time, which is one of the most important metrics when the detection should be in real-time. The camera resolution of the particular phone is 640x480, but for optimization purposes the application crops the images to equal dimensions according to the TF_OD_API_INPUT_SIZE variable. If it's set to 300, a 640x480 image will become, after the crop, a 300x300 image losing some information on both axes. After performing inference on several frames the inference time of the model is around 2350ms, which is not as real-time as it was hoped for. However, the application provides image recognition and tracking, which can be used to adjust the position and the size of the bounding boxes till the detections of the next frame arrive. Specifically, Image recognition and tracking detects the location of unique features in an image and attempts to find the locations of the exact same features in the next frame in order to calculate how much from frame to frame the unique features are moved and to which direction. This information is used to estimate how the mobile device has moved/rotated, which in consequence helps to adjust the bounding boxes from frame to frame. This process gives the illusion of real-time object detection in an attempt to compensate for the detections of the lost frames while an image was being processed. The following block diagram described the workflow of the application:


Application Launch
Permissions granted?
Yes
No
Ask for permissions
Yes
Permissions denied?
No
Start detecting objects in the current frame
Display new bounding boxes
Object detection finished?
Yes
No
Perform image recognition and tracking
Adjust position and size of displayed bounding boxes (if any)
Application Quit

Here are some images with their detected objects from the application:

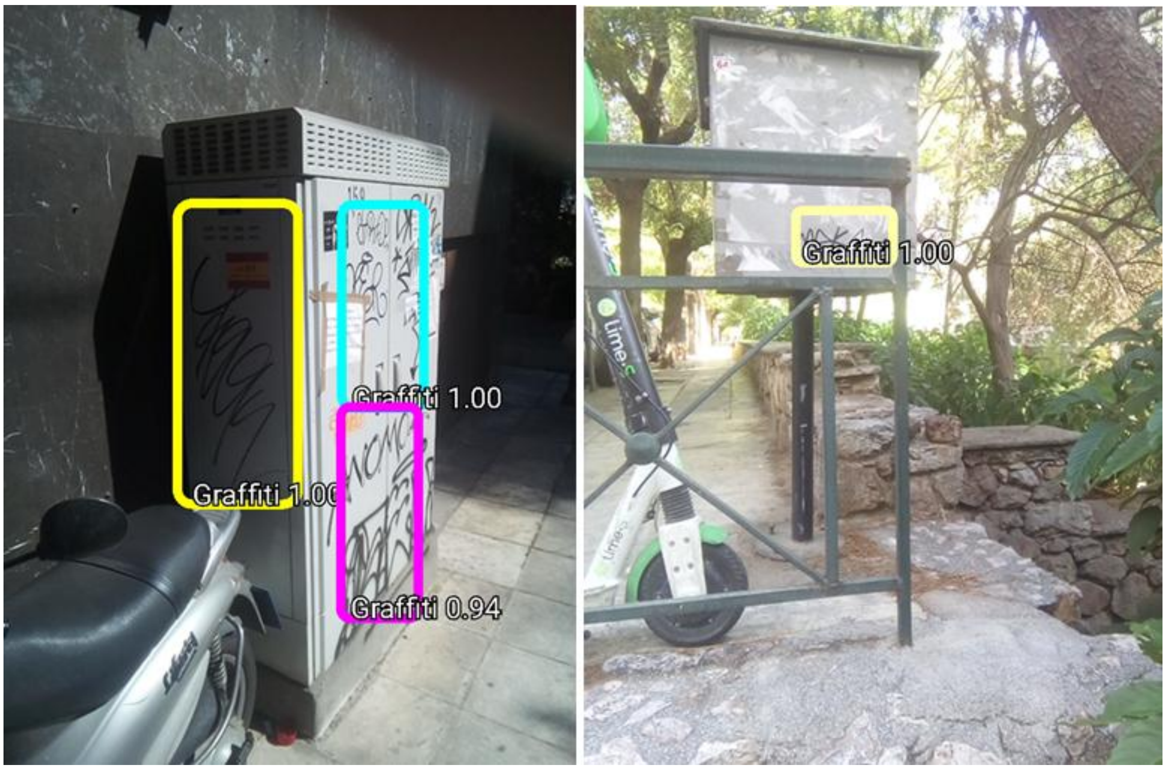


Figure 66: Images with detected graffiti tags on light-colored boxes

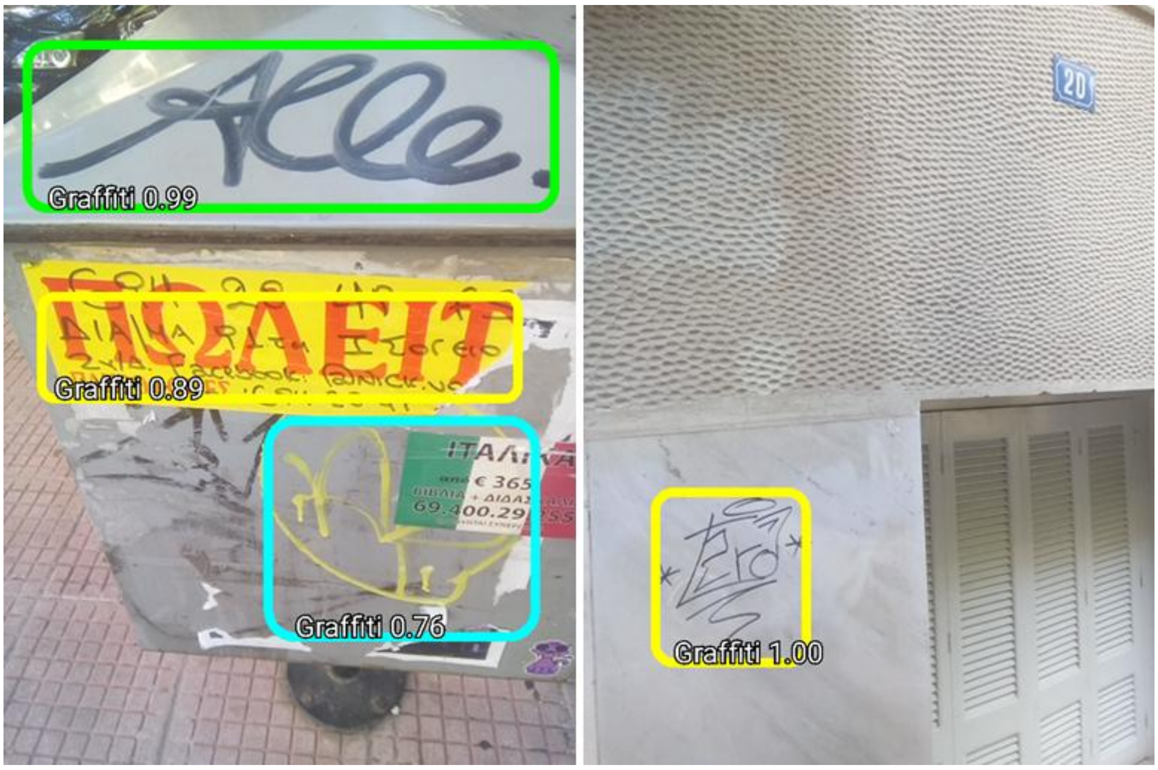


Figure 67: Detected most of the graffiti with false detection of hand-written text (left) and detected the single graffiti tag with no detection of the number (right)

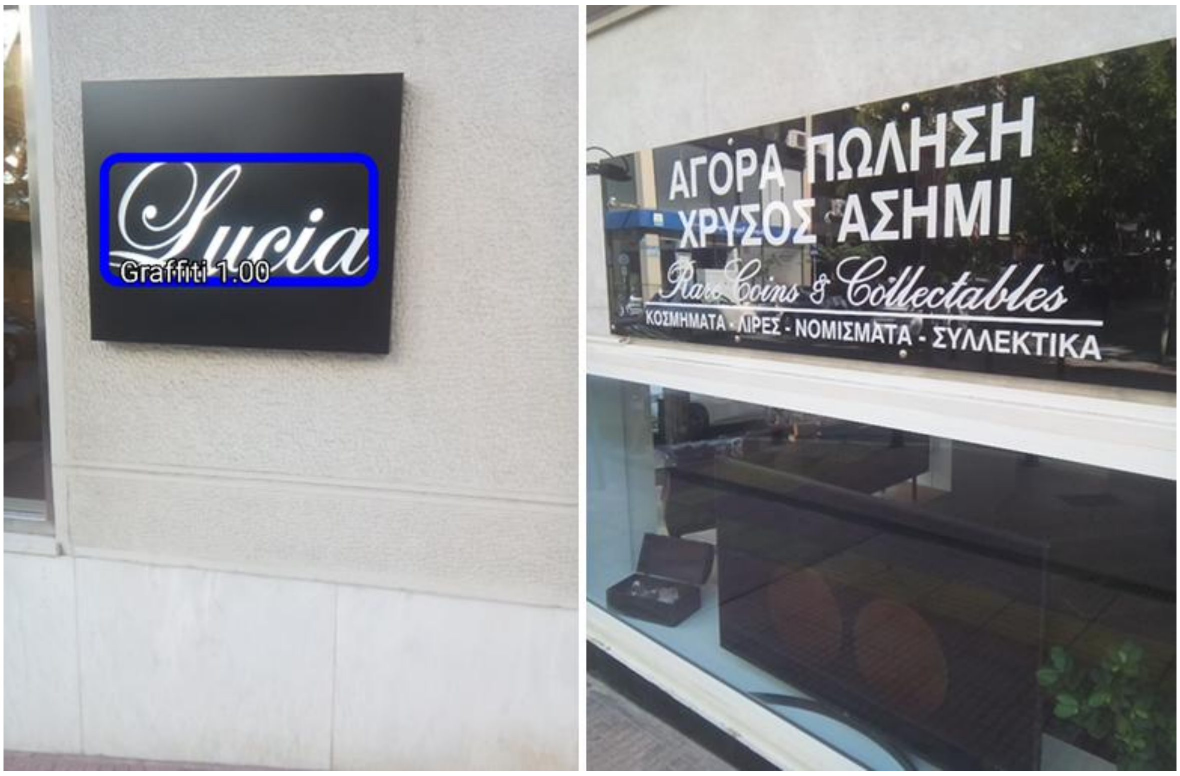

Lucia
Graffiti 1.00
ΑΓΟΡΑ ΠΩΛΗΣΗ
ΧΡΥΣΟΣ ΑΣΗΜΙ
Rare Coins & Collectables
ΚΟΣΜΗΜΑΤΑ - ΛΙΡΕΣ - ΝΟΜΙΣΜΑΤΑ - ΣΥΛΛΕΚΤΙΚΑ

Figure 68: Algorithm sometimes falsely detecting artistic printed text

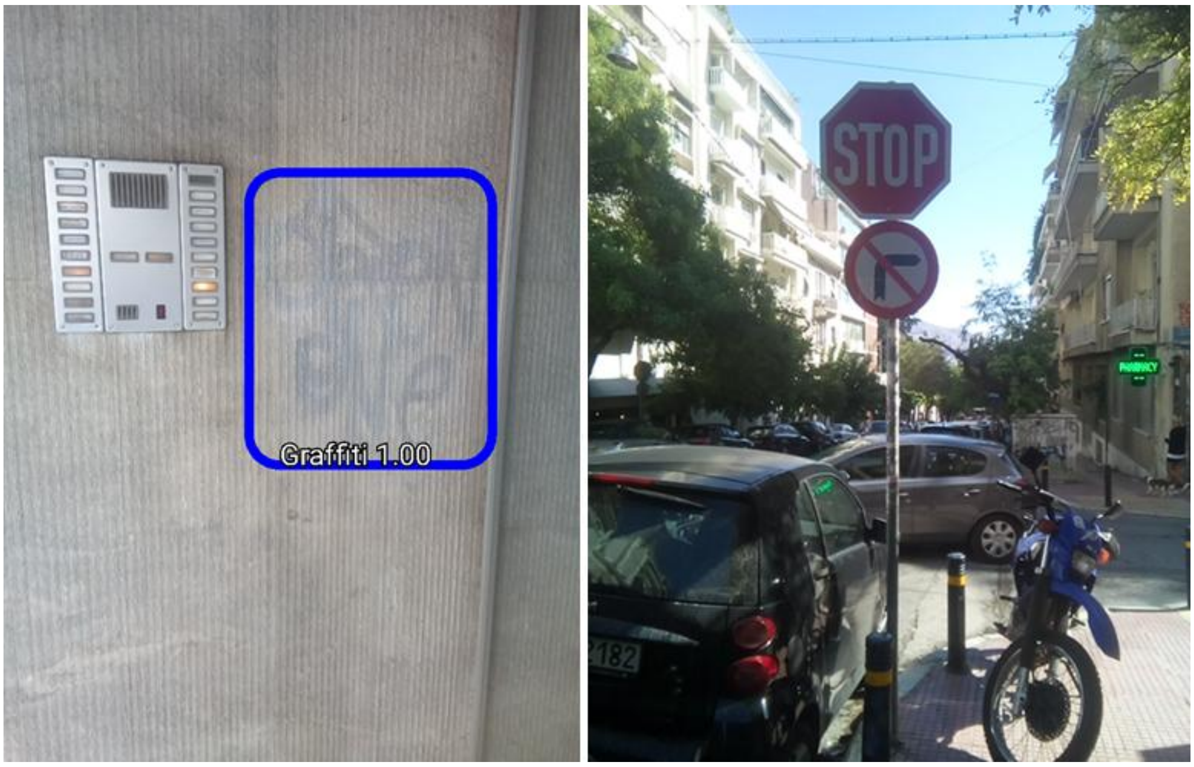


Figure 69: Algorithm correctly detecting faded graffiti tag (left) while also correctly not detecting shapes, such as traffic signs

Critical Discussion

In the making of this application, a lot of studying was conducted to understand how the underlying technologies work. As machine learning is a very complex field, most was comprehended in a sufficient degree, while other remained as a gray area. Though the development of the application, my knowledge of how a lot of commercial services function utilizing machine learning algorithms has vastly increased. Despite having a lot of obstacles throughout the development, the completion of a near real-time graffiti tag detection system was achieved.

Evaluation and Limitations

As mentioned before, the graffiti tag detection model has not achieved the best possible metrics. It has scored 0.53 using the F1 score, which derived from the precision and recall metrics, with 0 representing a failed model and 1 representing an excellent model having achieved a balance in the precision-recall trade-off. The most obvious reason that the model did not achieve a higher score is because it was purposed for a real-time application and as such, a compromise was made right from the beginning to get higher speed (smaller inference time) with the cost of accuracy. With other training algorithms, such as Inception or Resnet, the accuracy could easily increase from 0.6 to 0.7. However, the inference time increases exponentially. According to an experiment that was made out of curiosity using a COCO pre-trained Resnet model, although the accuracy was definitely higher, the inference time took over 10 minutes. Although this experiment was conducted on the same low-spec mobile device, which Resnet isn't designed for. Another possibility of increasing the performance of the model is to increase the size of the dataset by collecting and annotating more images, which greatly benefits the deep learning approach of this project. 2000 images is considered a small dataset. However, transfer learning was used to partially compensate for the size of the dataset, which is just enough to prove the concept of this project. Lastly, there could have been better management with the settings in the configuration file. From the tensorboard graphs, the loss seemed too noisy (before smoothing out the curve), which leads me to the thought that learning rate might have needed better adjustments when changing it after some checkpoints.

Another reason for not achieving greater accuracy is the nature of how a graffiti tag is presented, which is something beyond the control of the developer. As mentioned in the introduction, graffiti tags is a (vandalistic) form of art. In some images, it is shown that store names, often printed in an artistic way, is mistakenly classified as graffiti tags. Hand-written text, especially Chinese letters (though the dataset contained no Chinese text to train on), are also often falsely detected. It seems that graffiti tags have an abstract nature, which makes achieving high detection accuracy an even harder goal.

Improvements

Other improvements that could be made to the application to take it to the next step are:

- More images with more variations of graffiti are needed to achieve higher accuracy.
- Using more classes to detect different types of graffiti. There are several types of graffiti. Some are urban art, which are harmless, other are tags, which this application focuses on and are mainly performed by youngsters around schools, and other are gang-related to mark the territory. Each type has a different style and can be used to identify its type.
- Communication with server to send image and location to be able to crowdsource a large amount of information. That way people could utilize information from other users in other areas to display a color-coded map (according to the type of graffiti) to get a profile of what kind of graffiti vandalism each area suffers from.

Possible uses

The automatic graffiti detection has numerous use-cases, including:

- Electronics surveillance

  The graffiti tags often are committed on private property. Such property could be buildings of corporate companies, moving vehicles (such as trains and buses) or more importantly ancient monuments that represent that cultural heritage of the country. All these usually already have a surveillance system installed. These cameras could have an automatic graffiti detection application running to catch those that create the graffiti tags on the act by sending a signal of the cameral location to roaming security personnel each time a bounding box appears. In the case of trains, cameras can be installed on the entrance of the building, where they're being maintained, signaling the clean-up service which train has graffiti on its wagons on its arrival.
- Community surveillance

  Graffiti tags are often in highly-visible places, which makes it greatly beneficial to have community surveillance. Citizens of an area that wish to keep their neighborhood free from graffiti tags could use the application to take screenshot of them and if the model detects at least one, it will be able to send the image with its bounding boxes and the location (using the device's GPS) of where the image was taken from to a server, which a city clean-up service has access in order to send employees to those locations.

Both use-cases require communication with a server, which is easy to implement given it requires only one-way basic communication. Also, it should be noted that both examples do not prevent vandalism directly. They do prevent it indirectly in the sense that if people knew that automatic detection systems are in use, they would possibly not attempt to start tagging knowing that they would be interrupted or their work would be cleaned up by the next day.

References

Articles in Online Encyclopedia

[1] B. J. Copeland, “Artificial intelligence”, *Britannica*.
Available: https://www.britannica.com/ [Accessed: 3 February 2019]

[7] "Arthur Samuel", *Wikipedia*.
Available: https://en.wikipedia.org/wiki/Arthur_Samuel [Accessed: 1 March 2019]

[16] "Deep Blue (chess computer)", *Wikipedia*.
Available: https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer) [Accessed: 10 April 2019]

[41] Agor153, “K-nearest neighbors algorithm”, *Wikipedia*, January 2013.
Available: https://en.wikipedia.org/wiki/File:Map1NN.png [Accessed: 3 May 2019]

[43] “Cluster analysis”, *Wikipedia*.
Available: https://en.wikipedia.org/wiki/Cluster_analysis [Accessed: 3 May 2019]

[45] “Semi-supervised learning”, *Wikipedia*.
Available: https://en.wikipedia.org/wiki/Semi-supervised_learning [Accessed: 4 May 2019]

Professional Internet Sites

[3] E. Roberts, "Neural Networks History: The 1940's to the 1970's", *Department of Engineering: Computer Science, Stanford University*.
Available: https://cs.stanford.edu/people/eroberts/ [Accessed: 3 February 2019]

[4] E. Reingold, "Artificial Neural Networks Technology", *Department of Psychology, University of Toronto*.
Available: http://www2.psych.utoronto.ca/users/reingold/ [Accessed: 9 February 2019]

[5] J. McCarthy, "Arthur Samuel: Pioneer in Machine Learning", *Stanford University*.
Available: http://infolab.stanford.edu/pub/voy/museum/ [Accessed: 9 February 2019]

[8] J. A. N. Lee, "Arthur Lee Samuel", *IEEE Computer Society*.
Available: https://history.computer.org/pioneers/samuel.html [Accessed: 1 March 2019]

[11] "Stanford Cart", *Stanford University*.
Available: http://web.stanford.edu/~learnest/cart.htm [Accessed: 10 April 2019]

[27] M. Villano, "Predictive Data Can Reduce Emergency Room Wait Times", *Stanford Business*, October 2016.
Available: https://www.gsb.stanford.edu/ [Accessed: 17 April 2019]

[46] “What Is Deep Learning?”, *MathWorks*.
Available: https://www.mathworks.com/discovery/ [Accessed: 4 May 2019]

[48] “Convolutional Neural Network”, *MathWorks*.
Available: https://www.mathworks.com/solutions/ [Accessed: 4 May 2019]

[49] “Deep learning approach to train new models faster by using pretrained models”, *MathWorks*.
Available: https://www.mathworks.com/discovery/ [Accessed: 5 May 2019]

[50] “What Is Object Detection?”, *MathWorks*.
Available: https://www.mathworks.com/discovery/ [Accessed: 5 May 2019]

[52] “Object Detection in a Cluttered Scene Using Point Feature Matching”, *MathWorks*.
Available: https://www.mathworks.com/help/vision/ [Accessed: 10 May 2019]

General Internet Sites

[2] S. Singh, “Cousins of Artificial Intelligence”, *Towards Data Science*, May 2018.
Available: https://towardsdatascience.com/ [Accessed: 3 February 2019]

[6] A. A. Narvaez, "Arthur Samuel, 88, Pioneer Researcher In Computer Science", *The New York Times*, August 1990.
Available: https://www.nytimes.com/ [Accessed: 1 March 2019]

[9] "Perceptron and Adaline", *Learn Artificial Neural Networks*.
Available: https://www.learnartificialneuralnetworks.com/ [Accessed: 1 March 2019]

[10] T. Vanderbilt, "Autonomous Cars Through the Ages", *Wired*, February 2012.
Available: https://www.wired.com/ [Accessed: 10 April 2019]

[12] “1960 – Stanford Cart – (American)”, *Cyberneticzoo*, December 2009.
Available: http://cyberneticzoo.com/cyberneticanimals/ [Accessed: 10 April 2019]

[13] "Learning, Then Talking", *The New York Times*, August 1988.
Available: http://www.nytimes.com/ [Accessed: 10 April 2019]

[15] G. Walters, "Artificially Intelligent Investors Rack Up Massive Returns in Stock Market Study", *Seeker*, March 2017.
Available: http://www.seeker.com/ [Accessed: 10 April 2019]

[17] J. Gardner, “Deep Blue”, *Flickr*, June 2007.
Available: https://www.flickr.com/photos/22453761@N00/ [Accessed: 10 April 2019]

[18] "Computer Technology Helps Radiologists Spot Overlooked Small Breast Cancers", *Cancer Network*, October 2000.
Available: http://www.cancernetwork.com/ [Accessed: 12 April 2019]

[19] E. V. Buskirk, "How the Netflix Prize Was Won", *Wired*, September 2009.
Available: https://www.wired.com/ [Accessed: 12 April 2019]

[20] E. V. Buskirk, "Bellkor’s Pragmatic Chaos Wins $1 Million Netflix Prize by Mere Minutes", *Wired*, September 2009.
Available: https://www.wired.com/ [Accessed: 12 April 2019]

[21] J. Markoff, "Computer Wins on ‘Jeopardy!’: Trivial, It’s Not", *The New York Times*, February 2011.
Available: http://www.nytimes.com/ [Accessed: 13 April 2019]

[22] L. Eadicicco, “IBM Researcher: Fears Over Artificial Intelligence Are 'Overblown'”, *Time*, May 2016.
Available: https://time.com/ [Accessed: 13 April 2019]

[23] "Google’s Artificial Brain Learns to Find Cat Videos", *Wired*, June 2012.
Available: https://www.wired.com/ [Accessed: 13 April 2019]

[24] J. Condliffe, "Google's Artificial Brain Loves to Watch Cat Videos", *Gizmodo*, June 2012.
Available: http://gizmodo.com/5921296/ [Accessed: 13 April 2019]

[25] D. Aamoth, "Interview with Eugene Goostman, the Fake Kid Who Passed the Turing Test", *Time*, June 2014.
Available: http://time.com/ [Accessed: 13 April 2019]

[26] "Six Novel Machine Learning Applications", *Forbes*, January 2014.
Available: https://www.forbes.com/ [Accessed: 13 April 2019]

[28] "A Short History of Machine Learning -- Every Manager Should Read", *Forbes*, February 2016.

Available: https://www.forbes.com/ [Accessed: 2 May 2019]
[29] C. Koch, "How the Computer Beat the Go Master", *Scientific American*, March 2016.
Available: https://www.scientificamerican.com/ [Accessed: 2 May 2019]
[30] L. Morgan, "11 Cool Ways to Use Machine Learning", *Information Week*, April 2015.
Available: http://www.informationweek.com/Accessed: 2 May 2019]
[31] P. Crosman, "How PayPal Is Taking a Chance on AI to Fight Fraud", *American Banker*, September 2016.
Available: https://www.americanbanker.com/ [Accessed: 2 May 2019]
[32] J. Condliffe, "AI Has Beaten Humans at Lip-reading", *Technology Review*, November 2016.
Available: https://www.technologyreview.com/ [Accessed: 2 May 2019]
[33] "10 Ways Machine Learning Impacts Customer Experience", *Forbes*, February 2017.
Available: https://www.forbes.com/ [Accessed: 2 May 2019]
[34] M. Marshall, "The North Face to launch insanely smart Watson-powered mobile shopping app next month", *Venture Beat*, March 2016.
Available: https://venturebeat.com/ [Accessed: 2 May 2019]
[35] "The North Face Launches Mobile App Using IBM's Artificial Intelligence Watson", *Digital Retail Trend*, April 2016.
Available: http://digitalretail.co.kr/ [Accessed: 2 May 2019]
[36] "Now Anyone Can Deploy Google's Troll-Fighting AI", *Wired*, February 2017.
Available: https://www.wired.com/ [Accessed: 2 May 2019]
[37] J. J. Roberts, "Can Artificial Intelligence Silence Internet Trolls?", *Fortune*, January 2017.
Available: http://fortune.com/ [Accessed: 2 May 2019]
[38] A. Lockie, "This algorithm could help predict ISIS' future moves", *Business Insider*, September 2015.
Available: http://www.businessinsider.com/ [Accessed: 2 May 2019]
[39] "Machine Learning Explained: Understanding Supervised, Unsupervised, and Reinforcement Learning", *Datafloq*.
Available: https://datafloq.com/ [Accessed: 3 May 2019]
[40] R. Gandhi, "Support Vector Machine — Introduction to Machine Learning Algorithms", *Towards Data Science*, June 2018.
Available: https://towardsdatascience.com/ [Accessed: 3 May 2019]
[44] S. Ragate, N. Hmeidat, M. Aljumaily. "Pattern Recognition "Anomaly Detection Challenges"", *ResearchGate*, December 2015 .
Available: https://www.researchgate.net/ [Accessed: 3 May 2019]
[47] R. Parmar, "Training Deep Neural Networks", *Towards Data Science*, September 2018.
Available: https://towardsdatascience.com/ [Accessed: 4 May 2019]

Visual Media Sites
[42] Josh, "Decision tree, Machine learning, Big Data", *Pinterest*, March 2019.
Available: https://www.pinterest.com/ [Accessed: May 2019]
[51] S. Baig, "People Detection with HOG (Histogram of Oriented Gradients) and SVM (Support Vector Machines)", *YouTube*, August 2014.
Available: https://youtu.be/b_DSTuxdGDw?t=13 [Accessed: 5 May 2019]

E-books
[14] R. Chopra, "Advanced Computer Architecture".
Available: https://books.google.com/books?id=o58GiMl-_vkC&pg [Accessed: 10 April 2019]

Hardware
[56] GeForce GTX TITAN X
Available: https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x/specifications [Accessed: 10 February 2019]
[57] Turbo-X A2
Available: https://www.plaisio.gr/tilefonia-tablet/tilefona/smartphones/turbo-x-a-2-3g-smartphone-leuko_2581752 [Accessed: 10 February 2019]

Software
[53] TensorFlow
Available: https://www.tensorflow.org/install [Accessed: 10 February 2019]
[54] Android Studio
Available: https://developer.android.com/studio [Accessed: 10 February 2019]
[55] “TensorFlow Android Camera Demo”, *GitHub*. Available: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android [Accessed: 10 February 2019]
[58] “Raccoon Detector Dataset”, *GitHub*. Available: https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py [Accessed: 11 February 2019]
[59] “Tensorflow detection model zoo”, *GitHub*. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md [Accessed: 12 February 2019]
[60] “TensorFlow / Models”, *GitHub*. Available: https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs [Accessed: 12 February 2019]