Ville-Matti Mäyrä

Evaluating the sensitivity of lightweight object detection models against adversarial perturbations

Master's thesis of mathematical information technology

June 2, 2022

University of Jyväskylä

Faculty of Information Technology

Author: Ville-Matti Mäyrä

Contact information: ville.k.mayra@student.jyu.fi

Supervisors: Ilkka Pölönen and Oleksiy Khriyenko

Title: Evaluating the sensitivity of lightweight object detection models against adversarial perturbations

Työn nimi: Kevyiden hahmontunnistusmallien herkkyysanalyysi vihamielistä kohinaa vastaan

Project: Master's thesis

Study line: Software and Telecommunication Technology

Page count: 88+12

Abstract: The use of deep neural networks in the object detection task has become the mainstream solution in recent years due to the major improvements in the performance and accuracy of the detections that they have offered. Simultaneously the IoT technology has started to integrate with artificial intelligence technology and IoT devices with integrated cameras have started to adopt image recognition techniques. These have increased the interest in using object detection solutions on applications and devices in fields like surveillance and autonomous driving. Recently the applications have also started to adopt using deep learning models on-device to keep the unnecessary traffic off the network, which has supported the development of lightweight object detection models. However, the studies have shown that deep neural networks are vulnerable to adversarial examples, which are images that contain a subtle perturbation capable to fool the object detector to make false detection.

This thesis focuses to evaluate the sensitivity of lightweight object detection models against adversarial perturbation and studies what kind of adversarial attacks currently exist against object detection models. The experiments were conducted by crafting the adversarial examples using the python library designed to run adversarial attacks against deep neural networks. The evaluations were completed on selected pre-trained models while using

the datasets based on COCO 2017 dataset. Experiments show that using the adversarial perturbation crafted on the projected gradient descent method, the mean average precision of the selected models was decreased on low noise levels 4–10%, on average levels 10–28%, and on high levels 25–49%. When using adversarial perturbation crafted on the fast gradient sign method, the mean average precision of the selected models was decreased on low noise levels 10–22 %, on average levels 35–53%, and on high levels 70–84%. From the models used in experiments, the EfficientDet D0 512x512 model proved to resist the adversarial perturbation better than the others. Results show that pre-trained lightweight object detection models are vulnerable to adversarial examples crafted using the python library and it would need more research to make them more robust.

Keywords: Adversarial Examples, Deep Learning, Object Detection, Sensitivity Analysis

Suomenkielinen tiivistelmä: Syväoppivat neuroverkot ovat viime vuosina olleet yleisin käytetty menetelmä hahmontunnistuksessa niiden tarjotessa merkittäviä parannuksia suorituskykyyn sekä tunnistusten tarkkuuteen. Samanaikaisesti IoT-teknologia on alkanut integroitua tekoälyteknologian kanssa, jonka seurauksena kameroita sisältävät IoT-laitteet ovat alkaneet hyödyntämään kuvantunnistus tekniikoita. Nämä ovat kasvattaneet kiinnostusta hahmontunnistus ratkaisujen käytöstä sovelluksissa ja laitteissa eri aloilla, kuten valvonnassa ja itseohjautuvissa ajoneuvoissa. Viime aikoina syväoppimismalleja on implementoitu suoraan laitteisiin, tarpeettoman liikenteen pitämiseksi poissa verkosta. Tämä on tukenut kevyiden hahmontunnistusmallien kehitystä. Tutkimukset ovat kuitenkin osoittaneet, että syväoppivat neuroverkot ovat haavoittuvia vihamielisille esimerkeille, joilla tarkoitetaan vaikeasti havaittavaa kohinaa sisältäviä kuvia, jonka seurauksena hahmontunnistus mallit saadaan tekemään virheellisiä tunnistuksia.

Tämä tutkielma keskittyy toteuttamaan kevyiden hahmontunnistus mallien herkkyysanalyysin vihamielistä kohinaa kohtaan sekä tutkii millaisia vihamielisiä hyökkäyksiä hahmontunnistus malleja vastaan, on olemassa. Tutkimukset suoritettiin luomalla vihamielisiä esimerkkejä käyttäen python kirjastoa, joka oli luotu vihamielisten hyökkäysten testaamiseen syväoppivilla neuroverkoilla. Arvioinnit toteutettiin testaamalla valittuja esikoulutettuja malleja tietoaineistoilla, jotka pohjautuivat COCO 2017 tietoaineistoon. Kokeet osoit-

tavat, että projected gradient descent metodilla luodun hyökkäävän kohinan käyttö laski mallien keskimääräistä tarkkuutta matalilla kohinan tasoilla 4–10%, keskiverto tasoilla 10–28% sekä korkeilla tasoilla 25–49%. Käyttämällä fast gradient sign metodia hyökkäävän kohinan luonnissa, mallien keskimääräinen tarkkuus laski matalilla kohinan tasoilla 10–22%, keskiverto tasoilla 35–53% sekä korkeilla tasoilla 70–84%. Tutkimuksessa käytetyistä malleista parhaiten hyökkäävää kohinaa vastusti EfficientDet D0 512x512 malli. Tulokset osoittavat, että esikoulutetut kevyet hahmontunnistusmallit ovat haavoittuvia python kirjastolla luoduille hyökkääville esimerkeille ja tutkimusta mallien sitkeyden parantamiseksi tulisi jatkaa.

Avainsanat: Vihamielinen Esimerkki, Syväoppiminen, Hahmontunnistus, Herkkyysanalyysi

Glossary

AE	Adversarial Example
AP	Average Precision
CNN	Convolutional Neural Network
DNN	Deep Neural Network
EN	EfficientNet
FGSM	Fast Gradient Sign Method
GPU	Graphics Processing Unit
IoT	Internet of Things
mAP	Mean Average Precision
MN	MobileNet V2
PGD	Projected Gradient Descent
PSNR	Peak signal-to-noise ratio
RGB	Red Green Blue
RN	Resnet50
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machine
TF	TensorFlow
TF2	TensorFlow 2

List of Figures

List of Tables

Contents

1 Introduction

Object detection is one of the most fundamental and challenging problem areas in the computer vision field and it forms the foundations for solving complex or high-level computer vision tasks by detecting and locating pre-defined objects from natural images. While object detection has been an active area of research for several decades, the recent improvements in computational capabilities, availability of large-scale datasets, and use of deep learning techniques have provided major improvements in detection results. The deep learning neural networks have been a major force behind the remarkable breakthroughs made in the performance and accuracy of object detectors. The promising results have led to an increased interest in using object detection solutions in fields such as healthcare, security, agriculture, manufacturing, industrial and smart cities, while the object detection is already used to support applications related to robot vision, consumer electronics, surveillance and security, autonomous driving, human-computer interaction, content-based image retrieval, and augmented reality. (L. Liu et al., 2020.) Simultaneously the Internet of things (IoT) technology has been through rapid development in the past decade and has gradually started to integrate with artificial intelligence technology. One of the key technologies driving this integration forward has been image recognition, as video cameras are integrated into intelligent IoT equipment, like in autonomous driving. However, the use of deep learning solutions in IoT has raised questions regarding the security of the systems, due to recent findings made on deep learning architectures. (F. Wu et al., 2020.)

Studies from Szegedy et al. (2014) and Goodfellow et al. (2014) have demonstrated that deep neural networks are vulnerable to adversarial examples, which can cause serious security issues in applications. These adversarial examples are well-designed input samples, formed by adding small perturbations to the original sample. In object detection, the perturbations added to the image can be imperceptible to humans but can easily fool deep neural models to misclassify the input sample. While most of the studies around object detection focus on developing state-of-the-art deep learning models with better detection accuracies and faster processing times, the main concern should be pointed to the security issue created by vulnerability to adversarial examples as Y. Huang et al. state (2021). Eykholt et

al. (2017) showed that the traffic sign recognition systems can be misled to recognize a stop sign as a speed limit sign using adversarial examples, which can be very dangerous in autonomous driving applications. Simultaneously publicly available pre-trained object detection models are currently used in mobile applications while containing the same vulnerability against adversarial examples as Y. Huang et al. (2021) showed in their study.

Most of the studies related to the accuracy and vulnerability of object detection models have been concentrating on state-of-the-art object detection models. These models can be quite heavy and require a lot of computing resources to make the detections, which makes them unsuitable for IoT devices where computational capabilities are restricted. This creates a need for the study of how do the lightweight object detection models react to adversarial examples, which this thesis aims to answer. The main use cases for the lightweight object detection models are in the IoT devices where computation capacities are limited, or in situations where fast or real-time object detection is preferred over the high accuracy of the detections. In the IoT schema, the data is collected by the sensors on the device and usually processed either on the device or a backend server. In cases where the processing of the data can be made on-device, brings several advantages as the unnecessary traffic can be kept off the network. An example of this could be an IoT device that collects the video stream as an input through the embedded video camera and detects objects in real-time. The on-device deep learning models have gained attention recently as applications have started to adopt them (Y. Huang et al., 2021). This also supports the need for the study.

In this thesis, a sensitivity analysis will be conducted on selected lightweight object detection models when evaluating their detection accuracies on adversarial example inputs. The adversarial examples are crafted using the techniques already tested against the state-of-the-art deep image classification models. Selection of the lightweight object detection models was made by preferring the model applicability in the IoT devices.

This thesis aims to answer the following research questions:

1. What kind of adversarial attacks currently exists against object detection models?
2. How do the lightweight object detection models react to adversarial examples?

To answer these questions a literature review was concluded and experiments with adversarial examples were made. The experiments cover crafting the adversarial examples from selected images with proven methods and evaluating the accuracy of the pre-trained object detection models with adversarial datasets. The goal here was to solve how the selected models react to the adversarial perturbation included in images when doing sensitivity analysis, by comparing evaluation results with a clean dataset and with adversarially perturbed datasets. The crafting of adversarial examples follows the perspective that for humans, the difference between an original image and the adversarial one should be hard to detect, or the perturbation in the image could not be categorized as intentional.

This thesis is divided into nine chapters. The first chapter provides an introduction to the thesis. Chapters two, three, and four are part of the conducted literature review. Chapter two covers the topic of IoT from viewpoint of object detection, while chapter three explores the fundamentals of object detection, and chapter four builds an understanding of adversarial examples and their effects on object detection models. Chapter five covers the methodology used in the thesis. Chapter six demonstrates the process for running the experiments regarding the sensitivity analysis of lightweight object detection models against adversarial examples. In Chapter seven, the results of the experiments are presented. Chapter eight contains discussions about the results and limitations of this thesis work and presents some future research ideas. Finally, chapter nine concludes the thesis and summarizes the results. Appendix C contains information about copyrights and licences of images used in this research.

2 OBJECT DETECTION IN IOT DEVICES

This chapter covers the topic of object detection in IoT devices. The first subchapter includes a brief introduction and background of IoT devices. The second subchapter inspects the advantages and restrictions related to the use of deep learning in IoT devices. In the third subchapter, a few cases where object detection is currently utilized in IoT devices are presented shortly.

2.1 Introduction to IoT devices

The internet of things (IoT) is a concept that refers to the networked everyday physical devices, home appliances, automobiles, and other components with embedded intelligence, programs, sensor devices, and actuator controllers, that are interconnected to each other via wireless sensors attached to them and transfer useful information between devices. These devices use different techniques to collect useful data and automatically flow this to other devices and back. Internet of Things has gained growing attention from researchers and practitioners in recent years with aim of designing and developing complex IoT systems to meet the needs of the future smart world. (X. Li et al., 2011; Xia et al., 2012.)

One of the essential functions of IoT devices is the detection and recognition of the surrounding environment. The collected data may provide necessary information for the device itself or support for other functions included in the device. Deep learning-based object detection models play an important part in these kinds of devices, mainly because of their accuracy and efficiency in detecting objects. If the object detection model does not work properly it can directly cause problems in the IoT devices using the model, and eventually may cause serious consequences, especially if the device is used in critical operations like autonomous driving or security surveillance. Therefore, research about the security of object detection models has received more attention in recent years, and it should be a key feature to be improved in the future development of these models. (Wang, Tan, et al., 2020.)

2.2 Deep Learning and IoT devices

When deep learning applications are deployed on IoT devices they use one of the two major methods in how they function: the first is the **client-server** method and the second **client-only** method. In the client-server method, the IoT device is used as a sensor collecting the necessary data which is then sent to a cloud server running the deep learning model that processes the data and then sends the results to a client. This approach is good in situations where heavy computation can be done on a server. However, the key element in this approach is to maintain good connectivity between the device and the server. In the client-only method, the deep learning model is running directly on the IoT device where collected data is also processed. This way there is no need for a network connection, which may provide faster results, but the limitations of the device resources should be acknowledged. (Ota et al., 2017.)

As the use of artificial intelligence has become popular nowadays, deep learning methods have been included in mobile applications to support advanced functionalities like object detection. In many cases, the deep learning models are deployed onto a server, rendering the prediction to the application via the internet after taking the input from end devices. However, this approach has drawbacks such as severe latency, extensive server resource requirement, heavy bandwidth load, and privacy concerns. Therefore, many applications have started to adopt deep learning models on-device which is nowadays possible due to the development of computing capabilities on mobile devices. Deploying deep learning models on-device has several advantages. There is no round-trip to a server, which leads to bandwidth saving, faster inference time, and privacy-preserving as sensitive data stays on the device. Secondly, an internet connection is not required to run the application, which makes it more usable in rural areas. (Y. Huang et al., 2021.)

Most mobile devices such as wireless sensors, smartphones, tablets, and security cameras have limited resources in terms of available energy, computing power, memory, and network bandwidth. This makes it almost impossible to run state-of-the-art deep neural network models on-device because of the computing power and memory usage that they require to operate properly are not available. Earlier it was presented that one solution would

be the use of cloud computing, where powerful servers would handle the running of DNNs, but it would increase data traffic cost and has other drawbacks. Running top-end DNNs on-device would require either the capabilities of current mobile processors and chipsets to be increased or the complexity and memory requirements of DNN algorithms to be reduced. (Ota et al., 2017.) Chapter 3.4 presents some of the models that could be categorized as lightweight compared to top-end DNNs. These models are smaller in size and could be used on-device as they recognize the limitations of the computing capabilities of the IoT device.

2.3 Object detection applications in IoT devices

As L. Liu et al. (2020) stated in their article, the use cases for object detection can be found in applications related to robot vision, consumer electronics, security and surveillance, autonomous driving, augmented reality, human-computer interaction, and content-based image retrieval. While applications using object detection on-device might still be under development, or at least have not been released for public use, some of the existing applications have adopted object detection to provide additional information for the users.

For example, Adidas has created an application that allows customers to use their phone camera to seamlessly detect shoes in real-time, and within seconds returns an image recognition match from hundreds of products (Google, 2021b). Another example using object detection is Lookout by Google. It provides information about the surroundings of the user with a help of a phone camera and sensors of the device, as it recognizes objects and text. Lookout can be used to detect food by its packaging labels, scan documents to be magnified or read aloud by a screen reader, or skim the text and use text-to-speech to hear it aloud (Google, 2021a). Solutions for using object detection in agriculture were presented at the Tensorflow Developer Summit. Here the TensorFlow machine learning platform was successfully used by researchers at PlantVillage of Penn State University and the International Institute of Tropical Agriculture to help farmers in Africa to detect diseases in Cassava plants (Tensorflow, 2018).

3 FUNDAMENTALS OF OBJECT DETECTION

This chapter covers the fundamentals of object detection. The first subchapter offers a basic introduction to object detection and deep learning. In the second subchapter traditional object detection methods are briefly introduced before moving into deep learning-based methods. In the third subchapter, an operation of the neural network is presented. The fourth subchapter presents some of the backbone networks in object detection, and the fifth subchapter introduces a few popular object detection models.

3.1 Object detection and deep learning

Object detection is a computer vision task based on the object recognition field which combines the tasks of image classification and object localization. The purpose of object detection is to locate the presence of an object in an image, with a bounding box, and to classify these instances of objects to a correct class. The image classification task is responsible for predicting the type or class of an object and assigning a class label to it. The object localization task is responsible for locating the presence of objects in the image and indicating these with a bounding box around the object. Object detection combines these tasks in a way that first a bounding box is drawn around each object on the image, and then a class label is assigned to each bounding box as seen in Figure 1. (Brownlee, 2019.)

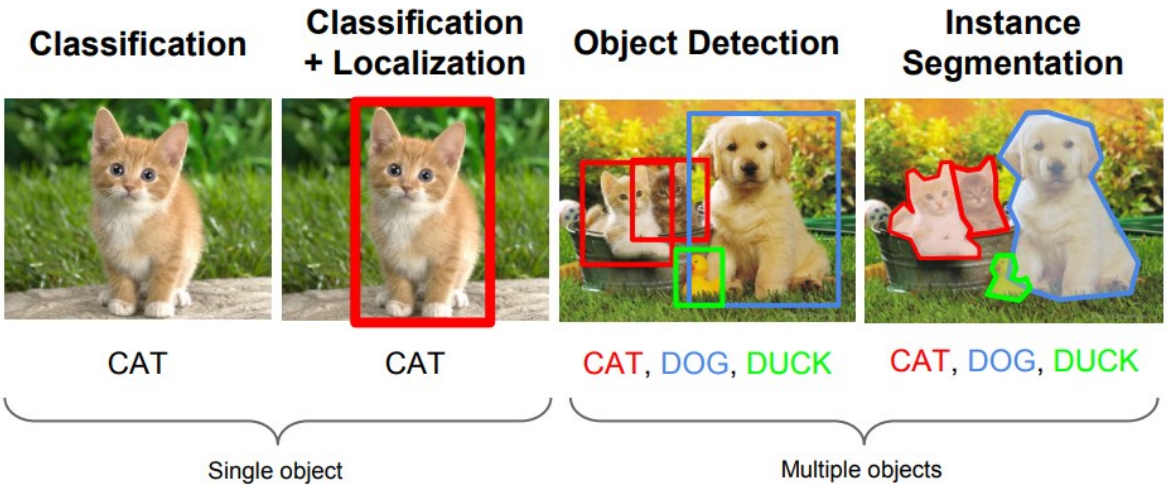


Figure 1. Computer vision tasks (F.-F. Li et al., 2016)

The approaches of object detection algorithms have been largely developed into two main categories: the traditional machine learning methods and the deep learning methods. The traditional machine learning-based object detection methods utilize a variety of computer vision techniques and rely on the established feature engineering for learning the features of the objects from a given input. The deep learning-based object detection methods in most cases use convolutional neural networks to capture and learn different features of the objects from given input and achieve better performance in large-scale data training. (Zou, 2019.)

In recent years the deep learning methods have been replacing the traditional machine vision methods because they have offered a better accuracy and performance in the same tasks. The history of modern deep learning can be divided into three development waves starting in the 1940s-1960s when it was referred to as cybernetics, the second wave in the 1980s-1990s known as connectionism, and the current wave under the name of deep learning which started in 2006. Cybernetics was based on the idea of biological learning, simulating the learning process of how human brains learn. At that time, the implementations of the first models such as the perceptron were created, which enabled the training of a single neuron. The wave of connectionism then introduced back-propagation to train a neural network with one or two hidden layers. The ongoing wave of deep learning started with a focus on new unsupervised learning techniques and the ability of deep models to generalize well from small datasets. However, the interest has since changed towards much older supervised learning algorithms and the ability of deep models to leverage large datasets containing labeled data. (Goodfellow et al., 2016.)

The current resurgence of deep learning has been possible due to two main trends: the increased amount of available training data, and the improved computational capabilities, which have made it possible to run much larger models and have helped deep learning models to grow in size (Goodfellow et al., 2016). Most of the current state-of-the-art deep learning-based object detection models are using a convolutional neural network architecture as their backbone. These models can be primarily divided into two families based on the way how they operate. One of the families includes two-stage region proposal-based detectors and other one-stage regression-based detectors. (X. Wu et al., 2020.)

3.2 Traditional object detection

Before deep learning-based models using convolutional neural networks became popular in the field of object detection, the traditional machine-learning-based methods were used in tasks such as pedestrian detection and facial detection. To determine the location and the class of an object in an image, the traditional object detection model pipeline can be mainly divided into three different stages including informative region selection, feature extraction, and classification of the object (Zhao et al., 2019).

The informative region selection stage is responsible for determining the possible locations of the objects in the input image. This is done by generating an initial set of candidate regions from the image, that further forms the potential object candidates. The set of regions should cover the majority of true object locations for it to be successful. This step can be seen as a rough classifier of whether certain regions contain objects of interest or not. To find all objects of different sizes, shapes, and positions, the simplest choice would be to scan the whole image using a multiscale sliding window. This method could find out all possible positions of the objects in a given image, but due to a large number of the possible object containing windows, it is computationally expensive and produces multiple redundant windows unnecessary for detection. Computational costs and the number of possible windows could be reduced by using a fixed number of sliding windows, but this might produce unsatisfactory regions. (Zhao et al., 2019.)

The feature extraction stage is responsible for recognizing different objects in the input image. This is done by extracting visual features that can provide a semantic and robust representation of objects. These features can be shapes, edges, or some other significant variation in RGB pixel values, that could refer to some object appearing in the image. In the traditional machine learning-based approach, these features are manually extracted. Representative features include scale-invariant feature transform, histograms of oriented gradients, and Haar-like. These features can produce representations associated with complex cells appearing in human brains, which makes them useful in object detection. However, because objects can appear in multiple angles and forms, and the lighting conditions

and background of the input image can vary, designing a robust feature descriptor manually is near impossible. (Zhao et al., 2019.)

Haar-like features can be thought of as a representation of a simple kernel in convolutional neural networks, where features are manually crafted instead of learned through training. Haar-like features can be used efficiently to detect edges and lines from the image. The main use cases for the features are in the field of face detection. The limitations of haar-like features in detections appear when edges and lines are unclear, or objects differ from basic type, e.g., eye, lip, or nose is covered by hand. Haar-like features do not require training because they are hand-crafted. They also need fever computations which makes the detections faster. (Lienhart & Maydt, 2002.) One of the most famous face detection frameworks that use Haar feature selection was created by Viola and Jones (2004).

The scale-invariant feature transform (SIFT) method transforms image data into scale-invariant coordinates relative to local features. The features are generated through four major stages. In the first stage scale-space extrema detection, the image is searched for locations over all scales to identify potential keypoints invariant to scale and orientation changes. In the second stage of keypoint localization, a detailed fit is performed to nearby data of keypoint candidates for accurate location, scale, and the ratio of principal curvatures. Acquired information allows rejection of the poor keypoints, like ones with low contrast and ones poorly localized along an edge. In the third stage orientation assignment, one or more orientations are assigned to each keypoint to achieve invariance to image rotation. Lastly in the fourth stage keypoint descriptor, a descriptor for the local image region is calculated to make SIFT more robust to the changes in object shape or lighting conditions. Through these steps, SIFT generates large numbers of features that densely cover the image over different scales and locations. A typical image of size 500x500 would produce about 2000 stable features by this method. (Lowe, 2004.)

Histogram of Oriented Gradient (HOG) is a feature descriptor, based on the idea that object shape can be effectively described by the distribution of edge directions and local intensity gradients. This can be done by extracting the gradient and orientation of the edges from the image. The image is divided into small spatial regions, and for each region gradi-

ents and orientation of the pixels are calculated. Then a histogram for each of these regions is generated to form the representation. To tackle the harmful effects of lighting conditions, the contrast-normalization techniques have been useful. (Dalal & Triggs, 2005.)

Working as the last section of the traditional object detection model pipeline, the classification of objects stage is responsible for distinguishing a target object from all other categories and making the representation of the objects more hierarchical, semantic, and informative for visual recognition. This is done by taking the extracted features and classifying the result into a certain class. In this stage, the supported vector machine (SVM), AdaBoost, and deformable part-based model (DPM) classifiers are usually good choices to go for. (Zhao et al., 2019.)

3.3 Deep Learning based object detection

Object detection has been gaining an increased amount of attention in recent years due to a wide range of applications and technological breakthroughs. There have been many factors that have made possible the fast evolution of object detection techniques, but the most notable ones are improvements in the computing power of GPUs and the development of deep convolutional neural networks. Most of the current state-of-the-art object detectors utilize the deep learning networks as a backbone for the model and a detection network for extracting features from the input image. (Jiao et al., 2019.)

Convolutional Neural Network (CNN) is a type of deep learning network that has become a popular approach in image classification and object detection tasks. After the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where CNN was applied in the image classification task with great success, it has become the dominant architecture in computer vision tasks. It provides an end-to-end learning model in which the gradient descent method can be used to train the parameters. Well-trained CNN can learn the features of the image more fully than using the feature extraction methods based on traditional object detection, while extracted features have a stronger ability of discrimination and generalization than the hand-crafted features. The features of a single layer are generated from the features of the previous layer's local area by sharing the weight of the convolutional

kernel. This makes the CNN more suitable for learning and representing the image features than other neural networks while keeping the translation and scale invariance in a certain scope. Typical CNN contains a convolutional layer, pooling layer, and fully-connected layer, but the development of object detection models has also created many new layers to tackle specific problems and to improve the performance of CNN. (Zhiqiang & Jun, 2017.)

The structure of the CNN architecture consists of an input layer and hundreds of feature detection layers which make the network deep before a classification layer determines the predictions for the output layer (Figure 2). Feature detection layers perform either Convolution, Rectified Linear Unit, or Pooling operation. Convolution is a linear operation where an image is put through convolution filters that activate certain features in the image. Convolution implements the multiplication of a set of weights with the input. Rectification Linear Unit (ReLU) is a linear activation function that maintains positive values and maps negative values to zero. Pooling performs non-linear downsampling to reduce the size of the input. In computer vision-related tasks, it means downsampling the detection of features in feature maps. The classification layer is located before the output layer. It is a fully connected layer with N-dimensional output, N being the number of classes to be categorized. The output of this layer is an N-dimensional vector where each element marks the probability of an input image belonging to one of those N classes. The final output layer is responsible for giving the classified output which is done with a softmax function. To obtain accurate results, the CNN needs to be trained with thousands or millions of images which usually requires multiple GPUs operating in parallel. (Vasuki & Govindaraju, 2017.)

In this thesis, it is beneficial to understand how the predictions are made on CNN. This is because the adversarial attacks aim to manipulate these predictions into something else than what they originally should be. The rest of the chapter describes the typical functioning of CNN, from taking the input image into a network to producing the output predictions.

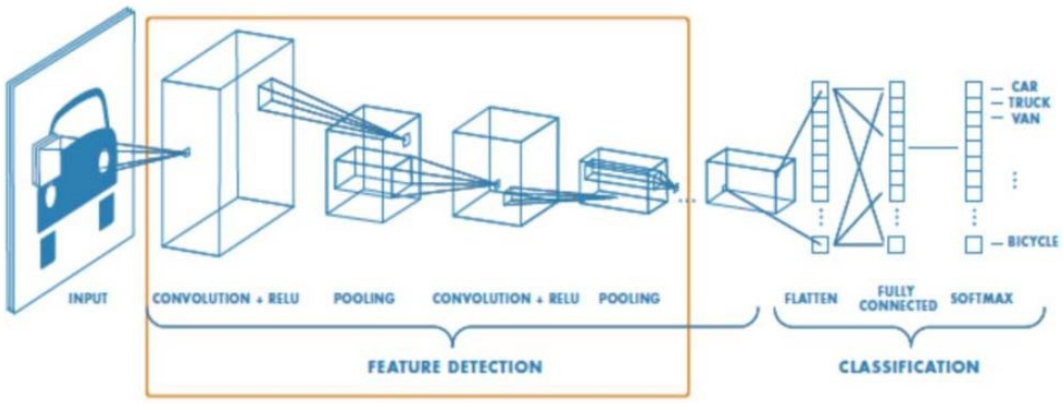


Figure 2. Convolutional Neural Network (Mathworks, n.d.)

Operation on CNN starts by passing an input image containing one or more objects to be classified into the network. The number of input values depends on the image size and the depth of the pixels, which in the case of the RGB image is 3. The RGB color image of size 256 x 256 can be represented as an array of 256 x 256 x 3, where all numbers must be translated or identified as objects belonging to a certain class. (Vasuki & Govindaraju, 2017.)

The first layer in CNN is a convolution layer responsible for performing a spatial convolution of a pre-defined mask with the pixel values. Equivalent to linear filtering operation, the result of convolution depends on the pre-defined mask which for the image of size 256 x 256 x 3 could be 5 x 5 x 3. In convolution, the pixel values and mask values are multiplied and summed up to get convolved output at the position of the mask placed over the image (Figure 3). The mask is then moved over the image from left to right and from top to bottom while repeating the convolution in each location until the whole image is covered. Mask values are determined by the shape to be detected. If the shape appears in that location of the image, the obtained convolved values are higher. Convolution produces a feature map which will be another array of numbers. A feature map works as an input to the second hidden layer where a similar convolution is repeated, and another feature map is produced. The second feature map is at a higher level than the first one. This process produces different activation maps for different features in the image and can represent even very complex features. (Vasuki & Govindaraju, 2017.)

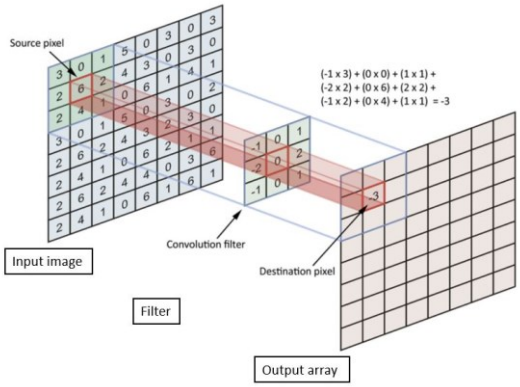


Figure 3. Convolution operation in CNN based on Vasuki & Govindaraju (2017)

The convolution layer is followed by an activation ReLU layer, making the operation non-linear. The negative activation values are turned to zeros and positive activation values are kept as they are. Activation layers are followed by pooling layers, such as the average pooling, stochastic pooling, overlapped pooling, or max pooling, that take the activation inputs and produce a downsampled output. Using the max pooling with a filter size of 2 x 2, the maximum value from the 2 x 2 activation region would be taken and the composition from all the regions would be the array output of this layer. To prevent overfitting of the model, dropout layers could be included in the architecture. These drop out certain activation outputs during the network training phase. (Vasuki & Govindaraju, 2017.)

At the end of the network, a fully connected layer takes its input from the preceding layer and produces N-dimensional output. In the N-class problem, this could output N different probability values which each represent the probabilities of the object belonging to that class. When using the softmax approach, the probability of an object belonging to a class would be between 0 and 1, where the higher rating would mean the most probable class for the object. (Vasuki & Govindaraju, 2017.)

3.4 Backbone networks in object detection

The architecture of the object detection model presents an overview of how the model is built to make detections from the given input. It is generally divided into two parts, the backbone network, and the head. The backbone network acts as the basic feature extractor

for the object detection task, taking the image as an input and producing the corresponding feature maps as an output (Jiao et al., 2019). The head part is responsible for making the detections from feature maps and usually contains either a one-stage or a two-stage detector to do so.

Most of the current backbone networks for object detection are based on the convolutional neural networks built for the image classification task without the fully connected layers. The backbone networks can be roughly divided between accuracy-oriented and efficiency-oriented networks depending on the use case of the model. Deeper and densely connected backbones like VGG16, ResNet, ResNeXt, and AmoebaNet, achieve better results in the accuracy of detections, but are complicated and contain a lot of parameters, which usually makes them unusable in mobile devices. Lightweight backbones like MobileNet, MobileNetV2, ShuffleNet, SqueezeNet, and Xception on the contrary, can meet the requirements of mobile devices and produce the detections efficiently, but do not achieve such detection accuracy as the larger backbones. In applications where real-time object detection from video or webcam is needed, the demands are usually high processing speed and high accuracy of detections. In these situations, a well-designed backbone is a key element that should adapt to detection architecture and make a trade-off between speed and accuracy. (Jiao et al., 2019.)

The interest in building small and efficient neural networks has been rising recently in the research literature. Approaches in this area can be roughly categorized either into methods that compress the pre-trained networks or methods that directly train small networks. One of the solutions that can help to implement deep learning models on-device is to reduce the size of the architecture of the model. One of the first examples of reducing the size of model architecture was made by Iandola et al. (2016) who introduced a small convolutional neural network called SqueezeNet. This lightweight model was able to achieve AlexNet-level accuracy on ImageNet while having 50 times fewer parameters and reducing the size of the model to less than 0.5MB, which was 510 times smaller than the original AlexNet.

Howard et al. (2017) introduced an efficient network architecture called MobileNets, that could be used to build very small and low latency models, suitable for mobile and embed-

ded vision applications. MobileNets architecture is based on depthwise separable convolutions. The deep learning models built on top of the MobileNet architecture are designed to effectively maximize the accuracy of detection while being aware of the resource restrictions of on-device or embedded applications. MobileNets differ from original large-scale state-of-the-art DNNs in a way that they are small, low-latency, low-power models that are parameterized to meet the resource constraints of a variety of use cases, like fast real-time detection. (Howard et al., 2017.) MobileNets approach was improved with MobileNetV2 by Sandler et al. (2018), who proposed using inverted residual block in architecture to achieve better performance. More recently Howard et al. (2019) introduced the third version of the MobileNet architecture family in form of MobileNetV3. This version contained large and small models demonstrating the new state-of-the-art approach in the mobile classification, detection, and segmentation tasks.

Another lightweight architecture approach was introduced by Zhang et al. (2018) in form of ShuffleNet, which used pointwise group convolutions to reduce computation complexity, and explored a channel shuffle operation to enhance the performance of lightweight models. ShuffleNet was further improved to the second version ShuffleNet V2 by Ma et al. (2018), who proposed that direct metrics such as speed should be considered over indirect metrics, like a number of float-point operations (FLOPs), when designing network architecture.

Han et al. (2019) presented a plug-and-play component Ghost module in their approach called GhostNet. The Ghost module can be used for building efficient neural architectures by reducing the computational costs of deep neural networks. The basic Ghost module is responsible for splitting the original convolutional layer into two parts and it utilizes fewer filters to generate several intrinsic feature maps. After this, a number of cheap transformation operations will be further applied for generating ghost feature maps efficiently. Due to the plug-and-play nature of the Ghost module, this component can be used to upgrade existing convolutional neural networks. (Han et al., 2019.)

3.5 Model head in object detection

The head part of the object detection model architecture is responsible for making detections from feature maps produced by the backbone. As mentioned earlier, the deep learning-based object detection models are mainly divided into two families based on how they operate. On one side are the two-stage region-based detectors and on the other one-stage regression-based detectors. In the following subchapters, both methods are briefly overviewed, and a few models are introduced.

3.5.1 Two-stage region proposal-based models

Two-stage object detection models divide the detection process into two stages. In the first stage, a sparse set of region proposals are generated from the image, and in the second stage feature extraction from the proposals is made followed by object classification and bounding-box regression tasks (Jiao et al., 2019). Figure 4 presents the basic architecture of two-stage detectors, consisting of a region proposal network that feeds the region proposals into a classifier and regressor.

In the region proposal generation phase, the object detector tries to identify regions from the image that potentially might contain an object. The detector should propose regions with a high recall and all objects in the image should belong at least to one of the proposed regions. During the second stage, region proposals are processed by the deep learning-based model which classifies the proposals with the right categorical label. These regions may either contain an object from predefined classes or belong to a background. The model might refine the original localization suggested by the proposal generator to make it more precise. Two-stage object detectors are often reported to get a state-of-the-art results with many public benchmark datasets. They are mainly more accurate than one-stage detectors but generally fall short in terms of lower inference speed. (X. Wu et al., 2020.)

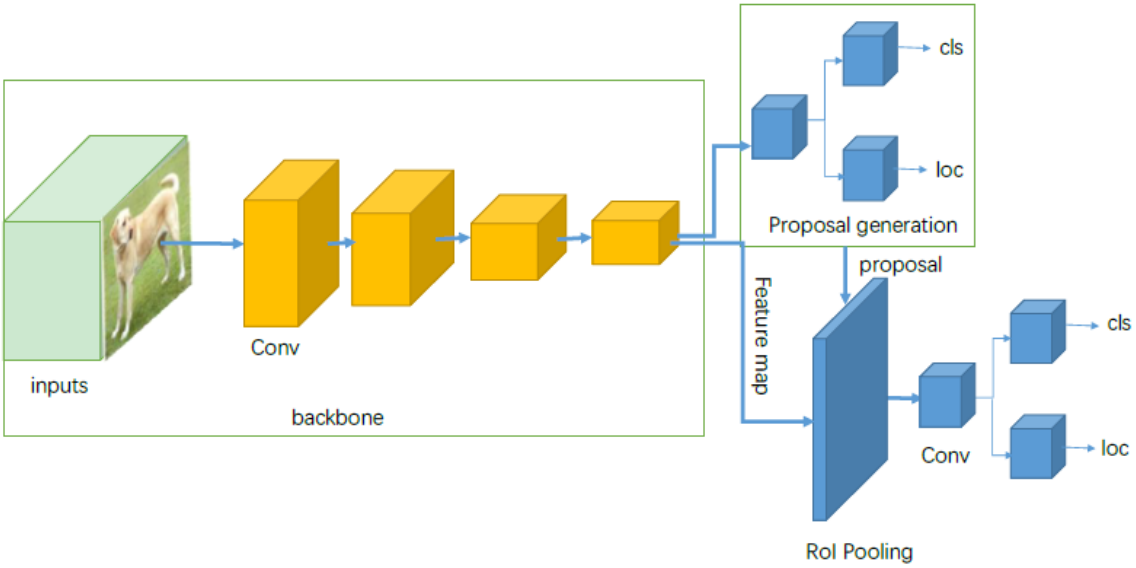


Figure 4. The basic architecture of two-stage detectors with RPN (Jiao et al., 2019)

Faster R-CNN (Ren et al., 2017) is an example of a very accurate region-based deep detection model that introduced Region Proposal Network (RPN) and combined it with Fast R-CNN (Girshick, 2015) object detector. RPN uses a fully convolutional network to simultaneously predict object bounds and objectness scores at each location, in order to solve the challenge of selecting the right regions for the second stage. In the second stage, the Fast R-CNN model takes the proposed regions from RPN as an input and provides the final object detection results. (Soviany & Ionescu, 2018.)

3.5.2 One-stage regression-based models

One-stage object detection models treat object detection as a simple regression problem and do not contain separate stages for the different tasks in the process. In these models, the region proposal and classification are performed simultaneously. All positions on the image are considered as potential objects, which are then tried to be classified either as a background or a target object. Most one-stage detectors follow the basic structure where the image is first spatially divided into a fixed number of grid cells, and then for each cell bounding boxes and class probabilities are predicted. One-stage detectors have dominated the two-stage detectors over the speed of the detections and are much more desired in real-time object detection applications. However, they achieve relatively poor accuracy com-

pared to two-stage detectors which have limited their usability in some cases. (X. Wu et al., 2020.) Figure 5 presents the basic architecture of one-stage detectors.

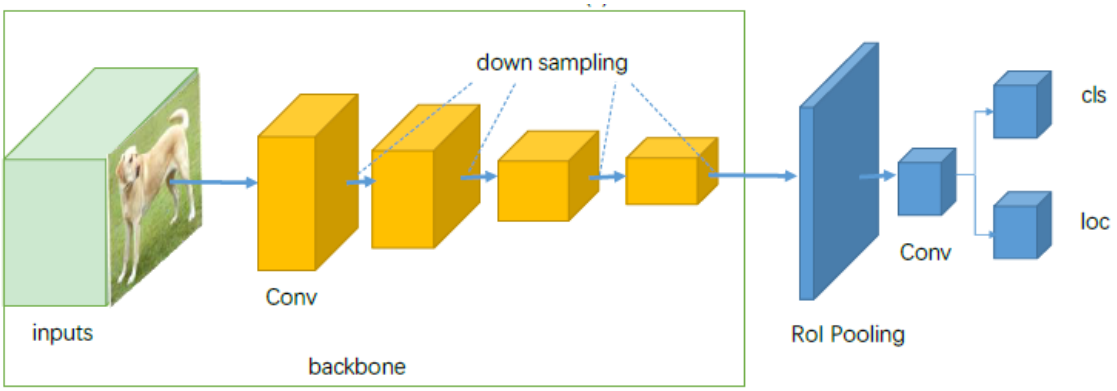


Figure 5. The basic architecture of one-stage detectors (Jiao et al., 2019)

YOLO (Redmon et al., 2015) object detection model is an example of a fast-performing one-stage regression-based model. It takes a full image as an input and uses a single neural network to predict the bounding boxes and corresponding classes. YOLO divides the image into a fixed size grid, and for each cell in a grid, predicts a number of bounding boxes and confidence for each bounding box. The confidence here reflects the accuracy of the bounding box and whether the bounding box actually contains an object regardless of its class. It also predicts the class probabilities for each predefined class per box. After the calculations, the bounding box and class probability are combined, and the object is found by using non-maximum suppression. (Soviany & Ionescu, 2018.)

SSD (W. Liu et al., 2015) object detection model is another example of a single-stage detector treating object detection as a simple regression problem. It uses a set of predefined boxes of different aspect ratios and scales to predict the presence of an object in the image. SSD encapsulates all computations in a single network and thus does not include separate proposal generation or resampling stages. (Soviany & Ionescu, 2018.) SSD combines the regression idea of the YOLO and the anchor method of the Faster R-CNN to make predictions from feature maps of different scales. The SSD model can achieve high detection accuracy of the Faster R-CNN model and the speed of the YOLO model, which makes it suitable for real-time object detection. (Zhiqiang & Jun, 2017.)

CenterNet (Duan et al., 2019) object detection model is a low-cost one-stage detector built upon the keypoint-based detector CornerNet. While CornerNet uses pair of keypoints to detect the top-left and bottom-right corners of the object to produce a bounding box, CenterNet uses a triplet that also includes the keypoint at the center of corner keypoints. CenterNet inherits partially the functionality of RoI pooling (Region of interest pooling) while still being recognized as a one-stage detector. The idea behind CenterNet is that when predicting the bounding box of the object, corner keypoints, and center keypoint should all belong to the same class. By exploring the visual patterns within the bounding box, CenterNet should reduce the number of incorrect bounding boxes around objects while improving the precision and recall of the model. (Duan et al., 2019.)

4 ADVERSARIAL EXAMPLES

This chapter covers the topic of adversarial examples from the viewpoint of computer vision tasks. The first subchapter offers the basic introduction to adversarial examples, which is then followed by a brief introduction to adversarial machine learning. Adversarial examples are then observed together with deep learning to provide a better understanding of the current situation. Summary of current defense methods against adversarial examples are introduced, and the related studies linking adversarial examples into object detection are also presented.

4.1 Introduction to adversarial examples in computer vision

According to Molnar (2019), an adversarial example can be described as an instance with small intentional feature perturbations that cause a machine learning model to make a false prediction. More commonly the adversarial examples can be thought to be a representation of optical illusion for machines. While many different techniques to create adversarial examples exist, most of the approaches include minimizing the distance between an adversarial example and the instance to be manipulated, while moving the prediction to desired adversarial result. In the case of neural networks, some of the methods require access to the gradients of the model, while others only need access to the prediction function, making these methods model-agnostic. (Molnar, 2019.) In the field of object recognition, the adversarial examples are images with intentionally perturbed pixels aiming to deceive the detection model during application time. The main purpose is to trick machine learning models to misclassify the input into something else that it originally is while closely resembling the original image as can be seen in Figure 6.

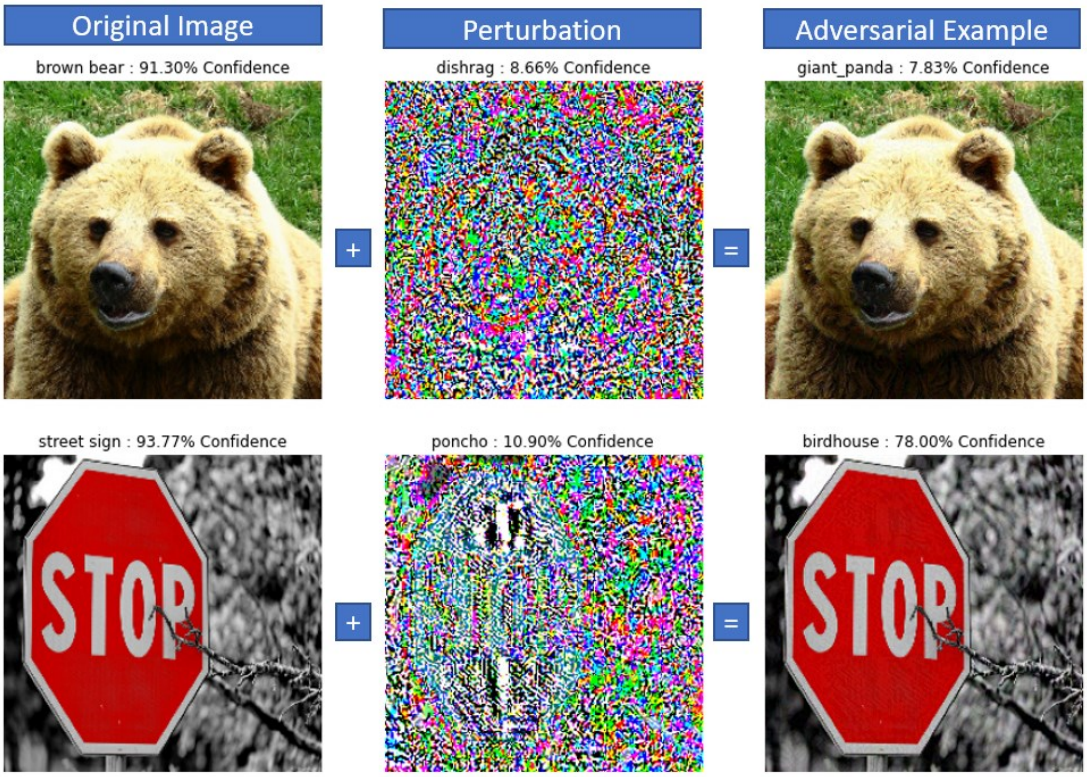


Figure 6. Adversarial example demonstration on images adapted from COCO dataset (Lin et al., 2017)

In a comprehensive survey about adversarial examples in object recognition, Serban et al. (2020) summed up adversarial examples in the following way: "Given a classification function **f** and a clean sample **x**, which gets correctly classified by **f** with label **y**, an adversarial example **x'** is constructed by applying the minimal perturbation **η** to input **x** such that **x'** gets classified with a different label **ŷ**."

$$\arg m\, in_{\eta} f(x+\eta) = \hat{y}$$

In case of the object detection, this main goal of adversarial examples can be further divided into different subgoals. These subgoals can include vanishing objects from the image, fabricating the image with additional false positives, targeted or untargeted mislabeling of the detected objects, and mislocalization of the detected object. Figure 7 shows example results of attack methods from the TOG suite developed by Chow et al. (2020).

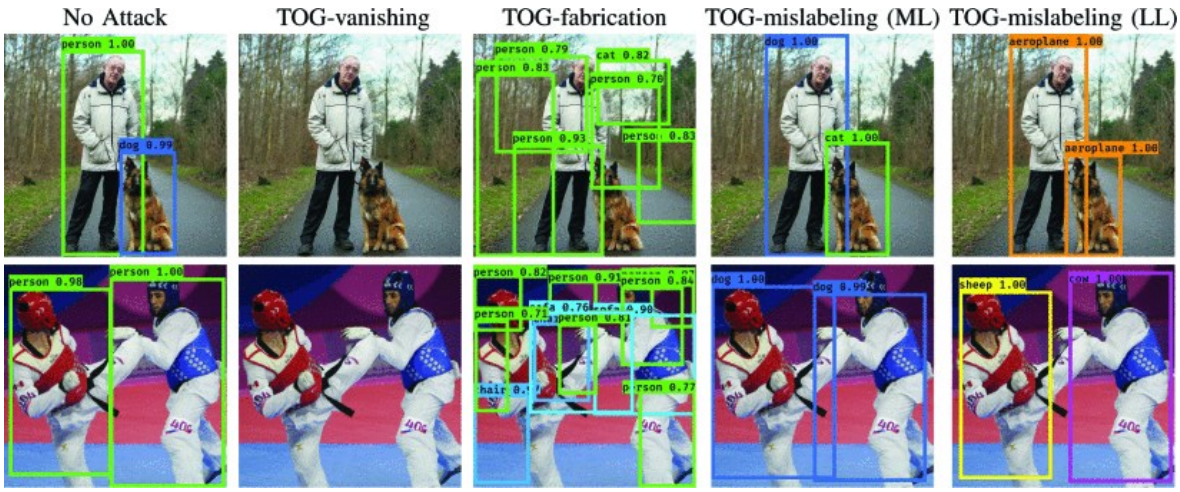


Figure 7. Visualization of TOG attacks in object detection (Chow et al., 2020)

One commonly known property among adversarial examples is their transferability between the detection models. Transferability means that the adversarial examples generated for one model may also be misclassified by another model trained to perform the same task. First studies of this property were made by Szegedy et al. (2014) and were continued by Goodfellow et al. (2014), Papernot, McDaniel, & Goodfellow (2016), and Papernot, McDaniel, Goodfellow, et al. (2016). (Y. Liu et al., 2016.)

The existence of adversarial examples in deep neural networks is still an open question and while many hypotheses have been proposed to explain the situation, there is no universal explanation as to why they exist. Yuan et al. (2019) suggested three different reasons behind adversarial examples by reviewing recent findings on the topic. The first reason was the **incomplete datasets**, which claims that the adversarial examples exist because of low probability and low test coverage of corner cases in datasets. The second reason referred to the **model capabilities**, where one claim was that the models are too linear in high-dimensional manifolds. The third reason was about the **robustness of the model**, where one example was that the decision boundaries of deep neural networks are inherently incorrect which causes the existence of adversarial examples. (Yuan et al., 2019.)

To understand adversarial examples, it is good to have some knowledge about the key terms used in the field. Akhtar & Mian (2018) described the common technical terms that

are used in literature when discussing adversarial attacks on deep learning in computer vision. This thesis follows the same definition of the following terms:

- **Adversarial Example (Adversarial Image)** is a modified version of a clean image that is intentionally perturbed to fool a deep neural network.
- **Adversarial perturbation** is the noise that is added to a clean image on purpose.
- **Black-box attack**, where adversarial examples are generated and inputted to the targeted model without any knowledge about the target model.
- **Non-targeted attack**, where the model is fooled to falsely predict any label for an adversarial image as long as it is not the correct one.
- **Targeted attack**, where the model is fooled to falsely predict a specific label for an adversarial image.
- **Threat model** refers to the type of potential attacks against the model, e.g., a black-box attack.
- **Transferability** refers to the ability of adversarial examples to remain effective against other models.
- **Universal perturbation** refers to the ability to fool a given model on any image with high probability.
- **White-box attack**, where an attacker has complete knowledge of the targeted model. Including its parameters, architecture, training method, and in some cases the training data.

4.2 Connections to adversarial machine learning

Publications and studies have shown that adversarial examples exist in fields where machine learning algorithms are used. The research field known as adversarial machine learning has gained renewed interest since deep networks for object recognition were shown to be vulnerable to perturbed images. This has increased the number of research papers proposing different types of countermeasures to mitigate the threat of adversarial examples in the field of computer vision but also fields like malware detection. The simplified categori-

zation of the main attacks against machine learning can be made between the test time evasion and training time poisoning as seen in Figure 8. (Biggio & Roli, 2018)

	Attacker's Goal: Misclassifications that do not compromise normal system operation	Attacker's Goal: Misclassifications that compromise normal system operation	Attacker's Goal: Querying strategies that reveal confidential information on the learning model or its users
Attacker's Capability	**Integrity**	**Availability**	**Privacy / Confidentiality**
Test data	Evasion (a.k.a. adversarial examples)	-	Model extraction / stealing and model inversion (a.k.a. hill-climbing attacks)
Training data	Poisoning (to allow subsequent intrusions) – e.g., backdoors or neural network trojans	Poisoning (to maximize classification error)	-

Figure 8. Evasion and poisoning attacks in machine learning (Biggio & Roli, 2018)

The development of adversarial examples can be observed through the history of evasion attacks. The first seminal work related to adversarial machine learning was done by Dalvi et al. (2004) and followed by Lowd & Meek (2005), who studied the problem with spam filtering and demonstrated that linear classifiers for spam filtering were vulnerable to adversarial examples. The linear classifiers could be easily tricked by creating a spam email with few changes from the original content without significantly affecting the readability of the spam message. The idea was to modify the first words with the highest weight values assigned by the linear text classifier. This was achieved by either changing bad words or adding good words into a message, or with a combination of these approaches. The heuristic countermeasures against these attacks were also proposed before 2010. The solutions would require learning linear classifiers with more uniform feature weights, which would require an attacker to make more changes to the content of the message to get it misclassified by the spam filter. (Biggio & Roli, 2018.)

Image spam was invented in 2005 to evade textual-based analysis. This resulted in developing the countermeasures like recognizing known spam images through hashing and extracting text from suspect images with optical character recognition tools. Attackers would then start to add random noise patterns to the spam images to evade these defense methods, which resulted in the use of learning-based approaches based on low-level visual features as a countermeasure. Srndic & Laskov (2013) showed in their work that evasion attacks

could be successfully done against linear PDF malware detectors such as decision trees and linear SVM. (Biggio & Roli, 2018.) Adding noise to the spam images and using the learning-based approaches as a countermeasure can be seen as a predecessor for current adversarial examples and defense methods in computer vision tasks, as the following chapters explain.

4.3 Adversarial examples against deep neural networks

While deep neural networks have become lately the mainstream solution to be used for solving complex problems in computer vision tasks, recent studies have shown their vulnerability to adversarial attacks. This poses a serious threat to the practical use of deep neural networks in current and future computer vision applications. Szegedy et al. (2014) first showed that adversarial examples, crafted by applying a certain hardly perceptible perturbation on top of the original image, can make a deep neural network on an image classification task to misclassify the image with high confidence. Goodfellow et al. (2014) argued that deep neural networks are vulnerable to adversarial examples because of the linear nature of neural networks and proposed a simple and fast method to generate adversarial examples in form of the Fast Gradient Sign Method (FGSM).

Moosavi-Dezfooli, Fawzi, & Frossard (2016) proposed an algorithm DeepFool, able to compute adversarial examples to fool state-of-the-art image classifiers. DeepFool computed the minimal adversarial perturbations that are sufficient to change classification labels, by assuming that the loss function can be linearized around the current data point at each iteration. In other research, Moosavi-Dezfooli, Fawzi, Fawzi, et al. (2016) showed the existence of universal adversarial perturbations, in which a small perturbation applied to any image was able to fool the image classifier. Baluja & Fischer (2017) trained feed-forward neural networks to generate adversarial examples against the target network or set of networks without using gradients. The Adversarial Transformation Network (ATN), was trained to generate adversarial examples that minimally modify the classifier's outputs given the original input so that the new classification result would match an adversarial target class. Kurakin et al. (2016) first demonstrated that adversarial examples also exist in the physical world. They used the mobile phone camera to obtain printed adversarial imag-

es which they used as an input to an ImageNet Inception v3 image classifier, with successful results of fooling the classifier to misclassify objects. Y. Liu et al. (2016) studied the transferability of non-targeted and targeted adversarial examples and developed an ensemble-based attack that successfully was able to fool a black-box image classification system in a targeted and non-targeted way. An extreme case of the adversarial example in deep learning neural networks was presented by Su et al. (2019) with One Pixel Attack. Their research showed that changing only one pixel of the input image was enough to successfully fool the image classifier to misclassify the input.

While most of the adversarial examples are meant to look like an original sample, some studies have shown the vulnerability of deep neural networks using a different approach. Nguyen et al. (2014) demonstrated in their research that it was easy to produce images that are completely unrecognizable to humans, but state-of-the-art DNNs believe these to be recognizable objects with high confidence. Brown et al. (2017) presented a method to create universal, robust, targeted adversarial image patches in the real world. They created a printable label that could be stuck next to objects in any scene and as a result, the other objects in the image would be ignored and the image classifier would classify the image as a toaster. This was the case regardless of the scale or location of the patch and did not require knowledge of other objects included in the image. Athalye et al. (2017) made a similar approach for 3D objects. They succeed in printing a 3D object in the shape and look of a turtle, while the deep learning 2D classifier detected a representation of a rifle from different angles and distances. Methods created by Brown et al. (2017) and Athalye et al. (2017) used the Expectation Over Transformation (EOT) algorithm, which generates adversarial examples that work even when the image is transformed. Results of these studies showed that adversarial examples and objects are a practical concern in real-world detection systems using deep neural networks, and that vulnerability could be exploited regardless of the viewing angle.

While most studies focus on the vulnerability of deep neural networks and how to exploit them, the studies about increasing the robustness of deep neural networks against adversarial examples have also been gaining increased attention in recent years. Madry et al. (2017) studied the gradient-based attack algorithm Projected Gradient Descent (PGD), where ad-

versarial examples are generated independently for each benign sample. They showed that adversarial training with the PGD adversary can provide better robustness against a range of other adversaries, meaning that adversarial examples would not have such an effect on the model. However, as the history of adversarial machine learning shows the defense methods often become inefficient when new attacks are invented. This has also happened already to some of the defense methods created against adversarial examples in deep neural networks. Carlini & Wagner (2016) presented three optimization-based algorithms that successfully created adversarial examples against neural networks using the defensive distillation method. They tested the attacks against image classification tasks and showed that while defensive distillation helps to increase robustness against adversarial examples generated with some methods, it did not prevent C&W attacks.

4.4 Defending against adversarial examples

As previously mentioned, the deep neural networks' vulnerability to adversarial examples has also increased the attention towards creating more robust deep learning models and implementing new defense techniques against adversarial examples. The literature does not contain a universal approach for grouping the defense methods against adversarial examples, but one popular way is to divide these between reactive and proactive methods. Serban et al. (2020) used this kind of approach in their survey calling the reactive ones as Guards and the proactive ones as Defense by Design. Defense methods under Guards do not interact with the model but interact instead with the adversarial input. Defense methods under the Defenses by Design category interact directly with the model by modifying model architecture, the training data, or the loss function before the model is deployed. These categories can be decomposed further as seen in Figure 9.

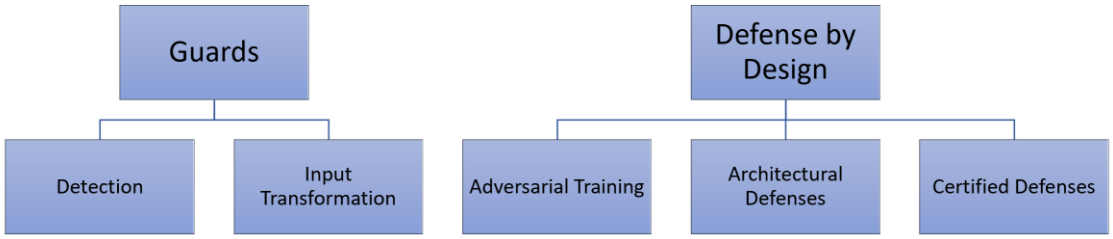


Figure 9. Defense strategies against adversarial examples based on Serban et al. (2020)

Detection methods rely on assumption that adversarial examples can be recognized from normal inputs. These methods usually include a separate detector trained to identify adversarial examples from normal inputs and either discard or skip the adversarial samples. While a wide range of adversarial detectors has been developed using very distinct features, they tend to fail in detecting strong adaptive attacks. Adversarial detectors are usually efficient against perturbations that they are trained to detect but need constant training against new attacks. This reduces their applicability as a defense method. **Input transformation** methods rely on pre-processing techniques such as compression and bit-depth reduction to remove the effect of adversarial perturbations and reduce their impact. These techniques are usually lightweight, low computational, and easy to implement and adopt. Input transformation methods often lead to stochastic gradients and offer good security against sensitivity analysis-based attacks. However, these methods are vulnerable to optimization-based attacks, which most times break the defense. Another downside is that transforming the input image might cause other problems, like loss of accuracy in the detections. (Serban et al., 2020.)

The **adversarial training** method is a regularization technique used to improve the robustness of neural networks by training them with adversarial examples. It relies on including adversarial examples as part of the training dataset. Adversarial training has shown very good results against adversarial examples in practice, and it has been mentioned as one of the state-of-the-art defense methods in the literature. **Architectural defense** methods propose to change the model's architecture by either imposing layer-wise constraints or by altering the final layer of the model. One example of architectural defense methods is distillation, where smaller DNNs are trained with knowledge from larger DNNs. This transfer

learning method has been seen to improve the DNNs robustness against adversarial examples. **Certified defense** methods rely on formal verification techniques and assume that within some bounds adversarial examples can not be found. These methods are useful in tasks where robustness is paramount, as they are able to guarantee that perturbations within certain bounds can not cause misclassification of the input. On the downside, some certified defense techniques are very complex, require more training resources, and are not yet scalable to deep models which limits their usability. Another downside is that guarantees can be made only to the training data and that some attacks can already bypass these defenses. (Serban et al., 2020.)

4.5 Adversarial examples in object detection

While most of the studies made around adversarial examples relate to image classification tasks, the practical computer vision applications are more often built around object detection which is a more complex topic. This has been acknowledged by the research community and studies related to adversarial examples in object detection have been gaining increased attention recently. This chapter introduces some of the studies around the topic to build an understanding of the current state of research and the findings that have been made. These findings will also be used to answer the first research question of the thesis.

Xie et al. (2017) extended the creation of adversarial examples from image classification to object detection and semantic segmentation based on the observation that both of the methods can be formulated into the task of classifying multiple targets on a given image. In object detection, the target is an object proposal. They proposed a Dense Adversary Generation (DAG) algorithm that optimizes a loss function over a set of proposals to generate adversarial examples. The suggested approach was able to efficiently fool object detection models with visually imperceptible perturbations and was shown to generalize well across different training sets, different network architectures, and even different tasks. (Xie et al., 2017.)

Addressing the DAG method's weak black-box attacking ability and the fact that its high computation cost made it unavailable for attacks on video object detection, Wei et al.

(2018) presented the Unified and Efficient Adversary (UEA) method for image and video object detection. This was claimed to be the first method capable of attacking effectively against image and video object detection tasks, while also being able to fool both region-proposal and regression-based object detectors. They utilized a generative mechanism instead of the optimization procedure to quickly generate adversarial images. UEA method was based on the Generative Adversarial Network (GAN) framework, combining a high-level class loss with a low-level feature loss to jointly train the adversarial example generator. To evaluate the efficiency of their attack they used the mAP (mean Average Precision) to compute the detection accuracy on the entire dataset before and after the attacks to see how much it would drop. UEA was shown to efficiently generate adversarial examples while being remarkably faster than previous attacking methods and having greater transferability between object detection models when tested with Faster-RCNN and SSD300 models. (Wei et al., 2018.) The same approach to evaluate detection accuracies and efficiency of the attacks is used in the experiments made in this thesis.

In their work, Y. Li et al. (2018) introduced a robust adversarial perturbation (R-AP) method for attacking deep proposal-based object detectors and instance segmentation algorithms. This method focused on attacking the Region Proposal Network (RPN) component of the model, which is used by the majority of object detectors, to extract region proposals from the image. They combined the label loss and the shape loss to calculate the adversarial perturbation. According to their best knowledge, it was the first work to investigate universal adversarial attacks against deep proposal-based models. Their evaluations with the MS COCO 2014 dataset showed that the R-AP was able to effectively attack several state-of-the-art object detectors. R-AP was also suitable for black-box attacking because the detectors rely on a few standard RPNs, and disturbing the RPN naturally degrades the deep learning model's performance. (Y. Li et al., 2018.)

Eykholt et al. (2017) proposed the algorithm Robust Physical Perturbations (RP_2) to generate perturbations that are robust to the change of distance or angle of the viewing camera. They used the algorithm to create adversarial examples for road sign recognition systems. Their study demonstrated that by applying crafted perturbation in a sticker or road sign poster and placing this onto a real road sign, the road sign recognition systems were suc-

cessfully fooled to detect stop signs as a speed limit. This approach was later extended from image classification to object detection in research made by Eykholt et al. (2018), where they also showed that it was possible to prevent physical objects to be detected from the image using the Disappearance Attack. They successfully evaluated this approach against state-of-the-art object detectors YOLOv2 and Faster R-CNN on video inputs.

Thys et al. (2019) presented an approach to generate adversarial patches that hide a person from a person detector. In their research, they used a printed patch or sticker to completely hide or lower the accuracy of detecting a person from the image when using the YOLOv2 object detector model in a real-world simulation. Another adversarial patch method was presented by X. Liu et al. (2018). Their black-box adversarial-patch-attack DPATCH applied a small, embedded patch in the input image causing Faster R-CNN and YOLO object detectors to not detect objects from the image. During the evaluations, this attack also demonstrated a great transferability between different architectures as the attack trained on Faster R-CNN was able to fool the YOLO object detector and vice versa.

Wang, Wang, et al. (2020) tested Project Gradient Descent (PGD) method to create adversarial examples against the Faster R-CNN object detection model with common backbone architecture. PGD method was stated to be the strongest first-order attack method on classification. Their tests showed that adversarial examples crafted with the PGD method were able to achieve a higher success rate than with the DAG method, which was rated as a state-of-the-art attack for generating adversarial examples against object detection models. While being more efficient and powerful than the DAG method, the adversarial examples crafted with the PGD method were not only misclassified but also mislocalized by the Faster R-CNN model. (Wang, Wang, et al., 2020.)

In their work, Chow et al. (2020) developed a suite of six different adversarial objectness gradient attacks (TOG) on deep object detection networks, with the capability to fool state-of-the-art object detection models either by untargeted or targeted attacks. The TOG-untargeted attack creates an adversarial example that can fool the object detector to randomly misdetect objects in the image without targeting any specific object, either by vanishing or mislabeling the objects, or providing multiple false detections. TOG-vanishing

attack removes the object detector's ability to identify objects from the image. TOG-fabrication attack fabricates the image with numerous false detections. The TOG-mislabeling attack makes the object detector misclassify the objects to the selected class. TOG-patch attack creates an adversarial patch that can either cause an object to be vanished or be mislabelled by the model. The TOG-universal attack creates universal adversarial perturbation by altering the pixel values of the entire image, which results in either vanishing objects from the image or fabricating the image with numerous false detections. The evaluations made with benchmark datasets VOC, COCO, and INRIA with state-of-the-art models YOLOv3, SSD, and Faster R-CNN showed that current object detection models contain serious adversarial vulnerabilities, which creates a compelling need for the development of robust object detection systems. (Chow et al., 2020.)

Another research made by Wang, Tan, et al. (2020) presented a black-box attack method against the region-based Faster R-CNN and regression-based YOLOv3 object detection models. They designed an optimization algorithm able to craft adversarial examples by only utilizing the position and label information of the predictions. Their Evaporate Attack was able to hide objects from being detected without any interior information of the model, reflecting a practical real-world case that attackers would face when attacking IoT systems using object detection. (Wang, Tan, et al., 2020)

5 METHODOLOGY

This chapter covers the methods used in this research. The goal of the research was to evaluate the sensitivity of lightweight object detection models against adversarial perturbation. For the research method, a sensitivity analysis was selected, as it is suitable for analyzing the changes that happen when clean and adversarially perturbed images are compared. As for selecting the lightweight object detection models, a set of pre-trained object detection models from Tensorflow Model Garden (Hongkun et al., 2020) were used. This is one of the most recognized repositories containing different implementations of state-of-the-art models for TensorFlow (Abadi et al., 2015). The repository contains TensorFlow 2 Detection Model Zoo which offers a collection of detection models pre-trained on the COCO 2017 dataset. TensorFlow Object Detection API (J. Huang et al., 2016) can be used to interact with these models. As for the dataset a Microsoft COCO 2017 validation dataset was selected to be used. This was chosen because the pre-trained models are trained with the full Microsoft COCO 2017 dataset and evaluated then with the validation dataset. As for the accuracy metrics, a COCO mAP (mean average precision) was selected from COCO detection metrics. The reason for this was that pre-trained models are evaluated using the same metric. Creating an adversarial dataset from COCO 2017 validation set and then evaluating models with this dataset, allows comparisons with COCO mAP metrics between datasets containing clean and adversarially perturbed images. The creation of adversarial examples was done with Foolbox (Rauber et al., 2017, 2020), which is a Python library made for testing the robustness of machine-learning models.

5.1 Sensitivity analysis

According to Allen (2017), a sensitivity analysis describes how susceptible a dependent variable is when a change occurs in a given independent variable. Sensitivity analysis is most commonly used with mathematical models of prediction, in which the independent variable is known as input and the dependent variable as output (Allen, 2017). In the scope of this research, the independent variable is the input image that the model is going to pro-

cess and the dependent variable will be the predictions that the object detection model produces.

Allen (2017) states that there is no one way how sensitivity analysis should be computed, but it depends on the problem area and variables used. With statistical tests, it is possible to determine what the actual observation of new values will be if the predicted value of the output has changed. In modeling, it is possible to build testable equations representing the observable phenomena for predicting the values estimated parameters by the model itself. Allen continues to state that relationships between variables are generally hypothesized in a such way that the input is said to be a cause of the observation of the output. Allen uses an example of annual income (Y) as a function of the number of years spent in school (X), which can be one way to describe the relationship between variables. (Allen, 2017.)

Allen (2017) also points out that sensitivity analysis is generally carried out with another statistical test known as uncertainty reduction. He states that the goal of the sensitivity analysis is to accurately determine how sensitive the observation of the output is, or how much the values change when the sample parameters change. In the case of the uncertainty analysis the same goal as with sensitivity analysis is accomplished; however, while sensitivity analysis puts a value on how susceptible to change the output is when input is changed, the uncertainty analysis generates a range of possible observations or values in the output based on the given set of input. Using the same example of annual income and years spent in school, the uncertainty analysis could be used to generate the actual range of possible observed values of the output. Sensitivity analysis using the same input and output would produce values that could illustrate how susceptible annual income is to changes in the parameters of years spent in school. Allen also states that once the most sensitive parameters have been discovered, it would prove beneficial to further investigate exactly which parameters are most useful to the prediction of the output. For a better understanding of the model and its power to predict the change in the output based on the input, it would be beneficial to use sensitivity analysis and uncertainty analysis together. (Allen, 2017.)

5.2 TensorFlow 2 Detection Model Zoo

TensorFlow Model Garden (Hongkun et al., 2020) is a GitHub repository containing pre-trained state-of-the-art deep learning and machine learning models and modeling solutions for TensorFlow (Abadi et al., 2015) users. TensorFlow models can be used as out-of-the-box implementation, or further fine-tuned for the specific task, through a TensorFlow 2 Object Detection API (J. Huang et al., 2016) or TensorFlow Hub API (Abadi et al., 2015). TensorFlow Hub provides a front-end website that can be used to search and discover models and datasets from across the TensorFlow ecosystem. Existing solutions for image problem domains vary from object detection to image classification, image segmentation, and image style transfer. At the time of writing, TensorFlow Hub shows 51 different object detection models trained on COCO 2017 dataset compatible with TensorFlow version 2 when TFLite models are excluded. TensorFlow 2 Detection Model Zoo however contains lists of 41 working models on object detection. The models that TensorFlow offers vary from deeper Faster R-CNN Inception ResNet V2 1024x1024 model to lightweight models such as SSD MobileNet V2 320x320.

In this research, the models are used through the TensorFlow 2 Object Detection API. Models that were selected from the TensorFlow 2 Detection Model Zoo for the sensitivity analysis were limited to having an interference speed of less than 40 ms. This limitation was chosen to comply with restrictions that using object detection in IoT devices might cause and because many IoT-related applications rely on fast detections over the accuracy. The selected models are pre-trained on COCO 2017 dataset. Models and their base metrics can be seen in Table 1.

Table 1. Selected models from TensorFlow 2 Detection Model Zoo with base metrics

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
EfficientDet D0 512x512	39	33.6	Boxes
SSD MobileNet V2 320x320	19	20.2	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22.2	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28.2	Boxes

A more informative description of the models can be found from TensorFlow Hub, with instructions on how to interact with them using the corresponding API. Models selected for this research are described in TensorFlow Hub in the following way:

- **EfficientDet D0 512x512** object detection model contains SSD architecture with EfficientNet-b0 backbone + BiFPN feature extractor, shared box predictor and focal loss. Trained with training images scaled to 512x512.
- **SSD MobileNet V2 320x320** object detection model contains SSD architecture, and it is trained on COCO 2017 dataset with training images scaled to 320x320.
- **SSD MobileNet V2 FPNLite 320x320** object detection model contains SSD architecture with FPN-lite feature extractor, shared box predictor, and focal loss. Trained with training images scaled to size 320x320.
- **SSD MobileNet V2 FPNLite 640x640** object detection model contains SSD architecture with FPN-lite feature extractor, shared box predictor, and focal loss. Trained with training images scaled to size 640x640.
- **CenterNet Resnet50 V2 512x512** object detection model contains CenterNet architecture with ResNet-v2-50 backbone. Trained with training images scaled to 512x512.

- **CenterNet Resnet50 V1 FPN 512x512** object detection model contains CenterNet architecture with ResNet-v1-50 backbone. Trained with training images scaled to 512x512.
- **CenterNet Resnet101 V1 FPN 512x512** object detection model contains CenterNet architecture with the ResNet-v1-101 backbone. Trained with training images scaled to 512x512.

5.3 Microsoft COCO 2017 dataset

Microsoft Common Objects in Context (MS COCO) is a large-scale image dataset of complex everyday scenes containing common objects in their natural context. The dataset was created to advance the state-of-the-art result in object recognition. MS COCO provides the dataset and evaluation metrics for different computer vision tasks, allowing users to evaluate their detection models on the dataset. The COCO 2017 dataset is divided into the training dataset, containing around 118 thousand images, the validation dataset containing 5000 images, and the test dataset containing around 41 thousand images. The dataset contains objects from 80 categories for object detection. An example image from COCO 2017 validation dataset can be seen in Figure 10. (Lin et al., 2014.)

Figure 10. Image based on COCO 2017 validation dataset (Lin et al., 2017)

The object detection models in TensorFlow 2 Detection Model Zoo have been pre-trained with the COCO 2017 dataset and their evaluation results are based on the mean average precision of COCO detection evaluation metrics. This made possible the use of the COCO 2017 dataset and COCO detection metrics in this research. Using the same evaluation technique to obtain the detection metrics in form of COCO mAP gave the possibility to use detection results presented in Table 1 when comparing the results between clean and adversarial datasets. For testing the sensitivity of lightweight object detection models, a validation part of the COCO 2017 dataset was downloaded from the internet. The COCO 2017 validation dataset contains 5000 images including objects from 80 different categories, and it is used to evaluate models trained with COCO 2017 dataset.

5.4 Foolbox – Adversarial attacks library

The crafting of adversarial examples was decided to be done using a popular Python library capable of generating the adversarial perturbation using different attack techniques and testing the robustness of machine learning models. Python library Foolbox (Rauber et al., 2017, 2020) was chosen for the task, while another library Adversarial Robustness Toolbox - ART (Nicolae et al., 2018) was also tested. Foolbox provides a large collection of state-of-the-art adversarial attacks to generate adversarial perturbation, and it is suited for comparing the robustness of models created with different frameworks. It has been cited over 220 times and has been used in numerous scientific publications, making it suitable for scientific research (Rauber et al., 2020).

The adversarial example crafting process in Foolbox requires five elements: First, a model that takes the input image and makes the predictions. Second, an adversarial criterion which states what adversarial should do. Third, a distance measure which measures the size of perturbation. Finally, an attack algorithm that takes the input, predicted label, machine learning model, the adversarial criterion, and distance measure to generate an adversarial perturbation. This perturbation can then be applied to the image making it an adversarial example. (Rauber et al., 2017.)

The following parameters were selected to be used in the experiments made in this thesis: For the input image, an image from COCO 2017 validation dataset is provided. For the machine learning model, a Tensorflow Keras model is passed from TensorFlow API (Abadi et al., 2015). These models contain canned architectures with pre-trained ImageNet weights and can be used for crafting adversarial examples for a classification task. Selected models are based on the common architectures, that are also used as backbones in pre-trained object detection models located in TensorFlow Model Zoo. For the adversarial criterion, a misclassification was selected. For the attack algorithm two gradient-based attacks from the Foolbox attack library were chosen, the Fast Gradient Sign Method (FGSM) and the Projected Gradient Descent (PGD). For the distance method in these attacks, L-Infinity Norm was used. The attack algorithms contain other configurable parameters that were kept as default when running the adversarial crafting function. As an output of this process a new image containing a certain amount of perturbation, depending on the selected epsilon value and other parameters, was produced.

The adversarial crafting process was run on each image in the validation dataset while changing the model, attack algorithm, and epsilon value parameters. This produced different compilations from the same image while the amount of perturbation in the image changed. The produced adversarial examples with the same parameters were combined as a new adversarial dataset. These datasets were then used to evaluate the robustness of pre-trained object detection models.

5.5 Peak signal-to-noise ratio in evaluating the level of perturbation

The gradient-based attack algorithms FGSM and PGD on the Foolbox attack library require an epsilon value to be selected in the adversarial crafting process. The epsilon value affects the amount of perturbation applied to the adversarial example. After testing different values with a few selected sample images from COCO 2017 validation set, the output images were first visualized and then used as an input for the pre-trained object detection model from TensorFlow 2 Detection Model Zoo via TensorFlow 2 Object Detection API on Google Colab. According to the prediction results and visualization of the adversarial examples, the epsilon values of 0.02, 0.05, and 0.1 were chosen to be presenting the differ-

ent stages of perturbation added to the image from low to high. These epsilon values were used with different model and attack algorithm parameters to craft adversarial datasets. Adversarial examples with the same epsilon value, model, and attack algorithm were combined as one new adversarial dataset. To further analyze the quality of the adversarial example and the amount of perturbation it contains, a more precise metric PSNR was calculated between a resized original image and a corresponding adversarial example. The average PSNR value was later calculated to full datasets for analyzing purposes.

Peak signal-to-noise ratio (PSNR) is used as a quality measure between the original image and a compressed image. PSNR values range from infinity to 0, where a lower value means a noisier image and a higher value means similarity to the original image. (Dixit & Phadke, 2013). In this research, PSNR is calculated between the resized version of the original image and the image containing adversarial perturbation, meaning it is used to calculate the level of noise that the crafted adversarial example contains compared to the original image. The high PSNR value means that the crafted image is looking like the original image and contains only a low amount of noise. The low PSNR value on the contrary would indicate a high numerical difference between the two images, meaning that perturbation added to the image is more visible and the image noise level is therefore higher. The typical values of PSNR in lossy image compression are between 30 dB to 50 dB (Prabhakar et al., 2011). PSNR can also be described as a rough metric to describe human visual perception when comparing the quality of two images (Sara et al., 2019).

5.6 COCO Metrics for evaluating the sensitivity

The COCO detection evaluation metrics were used to evaluate the sensitivity of lightweight object detection models against adversarially perturbed datasets. From all the available metrics this thesis focuses on Mean Average Precision (mAP), also stated as the COCO primary challenge metric. This metric was selected for two reasons: since most of the latest research papers are using it to evaluate the accuracy of the object detection models and because the accuracy of the object detection models in Tensorflow 2 Detection Model Zoo is presented using the same metric. A full list of COCO detection evaluation metrics can be seen in Figure 11.

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

Figure 11. COCO detection evaluation metrics (Lin et al., 2014)

COCO mAP uses a 101-point interpolated AP definition in the calculation. This means that interpolation is performed in N=101 recall points. Then computed results for each class are summed up and divided by the number of classes, making it an average across all categories. The COCO AP metric, also known as the primary challenge metric, computes the average precision with ten different IoU values from 0.5 to 0.95 with a step size of 0.05 and takes the average among all computed results. (Hui, 2018.)

The average precision (AP) metric evaluates a precision and a recall in different confidence values. In object detection, precision is used to measure how accurate the prediction is, meaning what percentage of the predictions are correct, while recall moreover measures how well can a model find all the positives. Figure 12 presents the mathematical definitions behind calculating precision and recall. (Hui, 2018.)

The third metric needed to calculate the AP is Intersection over Union (IoU), which measures the overlap between two boundaries. In object detection, IoU is used to measure the overlap between the predicted boundary and the ground truth boundary. IoU threshold classifies each prediction either as a True Positive (TP) or False Positive (FP). The threshold value can variate, but the starting point is usually 0.5 meaning that boundaries overlap 50%. Choosing a more restrictive IoU threshold e.g., 0.75 results a different precision and recall values which affect the AP results as seen in Figure 13. (Padilla et al., 2021.)

TP = True Positive ***FP = False Positive***
TN = True Negative ***FN = False Negative***

$$Recall = \frac{TP}{TP + FN} \qquad Precision = \frac{TP}{TP + FP}$$

Figure 12. Precision-Recall formulas based on Hui (2018)

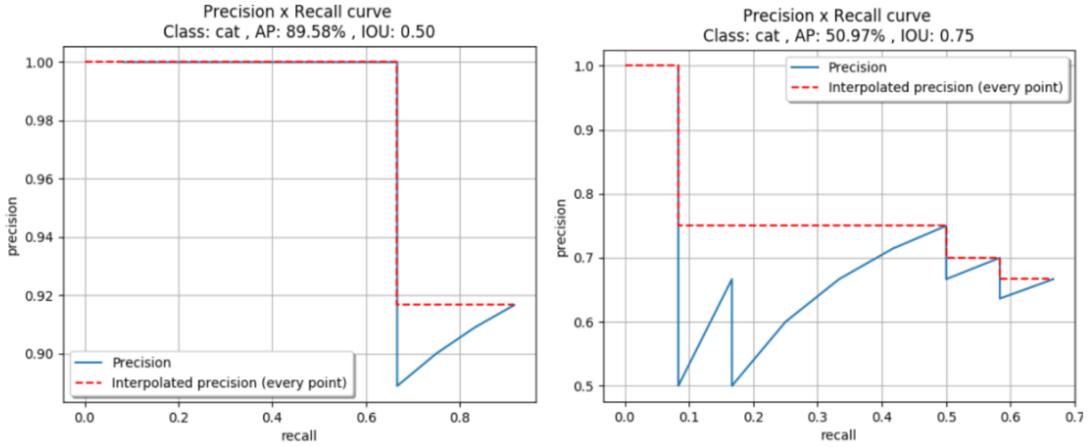


Figure 13. Average precision on different IoU (Padilla et al., 2021)

A similar approach to evaluate the performance of adversarial attacks in object detection by comparing mean average precision results obtained with a clean and adversarial dataset was used by Chow et al. (2020) and Wei et al. (2018) in their respective research. For the rest of the thesis when AP is used in the text, it refers to COCO primary challenge metric. The TensorFlow 2 Object Detection API (J. Huang et al., 2016) provides model_main_tf2.py script for creating and running TF2 object detection models, which can be used to calculate the mAP metrics of the model when evaluated with the dataset. Figure 14 shows an example of the model's COCO Detection metrics produced with the script on evaluation notebook. As part of the experiments, the script was run with each dataset and model to get metrics for comparing the effects that adversarial perturbations cause to the object detection model's accuracy. The evaluation runs were completed using the cloud-based platform Google Colab due to the insufficient number of local resources.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.136
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.234
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.137
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.025
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.179
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.363
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.168
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.258
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.272
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.084
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.397
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.606
```

Figure 14. COCO detection evaluation metrics produced with model_main_tf2.py script on evaluation notebook

6 EXPERIMENTS

This chapter presents the pipeline created for the experiments in this research and explains the selections made during the process. First, the selection of model architecture and attack algorithm parameters for the crafting process of adversarial examples are presented. Then decisions and experiments behind epsilon values used to craft adversarial examples are introduced. These three parameters affect the overall amount of perturbation included in adversarial examples and form the basis for the experiments. The third part introduces the creation of the datasets used in the experiments. The last part introduces the model evaluation process and visualization of the detections. Figure 15 demonstrates the process from crafting adversarial examples to running the evaluations. The pipeline and Jupyter Notebook files mentioned in the subchapters are available on the GitHub repository of this project (Mäyrä, 2022).

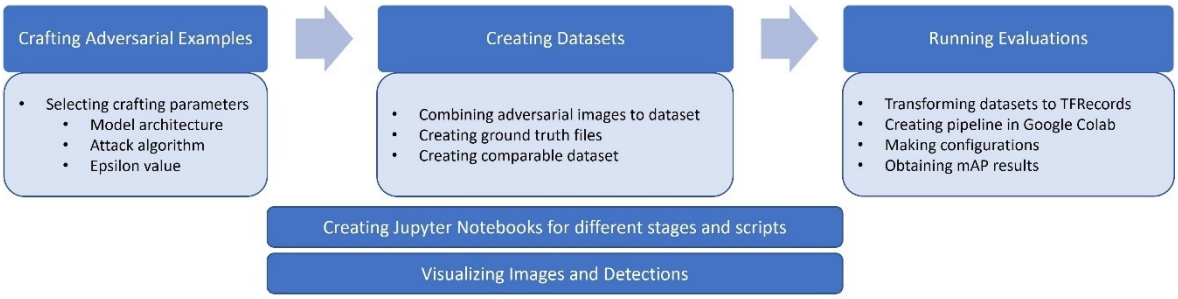


Figure 15. Flowchart demonstrating experiment pipeline of the thesis

6.1 Selecting model and attack parameters on Foolbox

The selection of the model parameter for crafting adversarial examples was based on the architectures present in the object detection models in this research. These architectures included MobileNetV2, Resnet50, and EfficientNetB0. Resnet101 architecture was present in one of the object detection models in this research but was not selected in crafting parameters because one Resnet family architecture was already involved. From the Foolbox library, the gradient-based attacks PGD and FGSM were chosen to be used as the attack algorithm parameter for crafting. These attacks were chosen after testing multiple different attacks from the Foolbox library and visualizing the crafted perturbations. The selection of

these attacks was based on two reasons: perturbation created by these attacks could be visualized clearly, and crafting adversarial examples with these methods was cost-efficient, which made it possible to craft many examples locally in a sufficient time.

6.1.1 Model Architectures

MobileNetV2 is a convolutional neural network architecture designed to be used on mobile devices for object detection and image classification tasks. It is based on an inverted residual structure where the residual connections are between the bottleneck layers. The intermediate expansion layer uses lightweight depthwise convolutions to filter features as a source of non-linearity. The fully built MobileNetV2 architecture contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers. Figure 16 shows the MobileNetV2 architecture and two types of convolutional blocks that are used in it. (Sandler et al., 2018.)

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

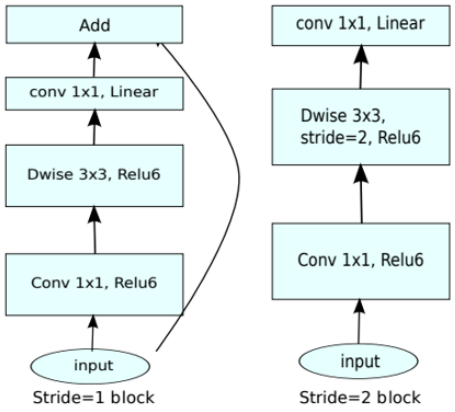


Figure 16. MobileNetV2 architecture and convolutional blocks (Sandler et al., 2018)

ResNet50 is a convolutional neural network architecture based on residual neural networks. ResNet50 is 50 layers deep and uses bottleneck building blocks in its architecture as seen in Figure 17. ResNet50 architecture is based on the original ResNet-34 architecture, but each 2-layer block is replaced by a 3-layer bottleneck block which results in the size of a 50-layer ResNet. Residual networks learn residual functions with reference to the

layer inputs, instead of learning unreferenced functions. This makes residual networks easier to optimize and gain accuracy from considerably increased depth. (He et al., 2016.)

layer name	output size	50-layer
conv1	112×112	7×7, 64, stride 2
conv2_x	56×56	3×3 max pool, stride 2 [1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[1×1, 128 3×3, 128 1×1, 512] ×4
conv4_x	14×14	[1×1, 256 3×3, 256 1×1, 1024] ×6
conv5_x	7×7	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax
FLOPs		3.8×10^9

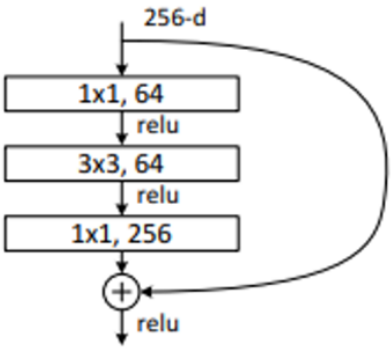


Figure 17. ResNet50 architecture and bottleneck building block (He et al., 2016)

EfficientNet-B0 is a mobile-size baseline architecture that uses mobile inverted bottleneck MBConv including squeeze-and-excitation optimization as its main building block. It is a part of EfficientNet convolutional neural network architecture and a compound scaling method. The EfficientNet scaling method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients, which makes it useful despite the size of the image that is used. Figure 18 shows the baseline architecture of EfficientNet-B0 (Tan & Le, 2019.)

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 18. EfficientNet-B0 baseline network (Tan & Le, 2019)

6.1.2 Attack Algorithms

The *Fast Gradient Sign Method* (FGSM) created by Goodfellow et al. (2014), is an attack algorithm that uses the gradient of the underlying model to find an adversarial example. It takes the original image and manipulates it by adding or subtracting a small error to each pixel. The deciding factor between adding or subtracting pixels depends on whether the sign of the gradient for a pixel is positive or negative. In other words, the idea is to adjust the input data to maximize the loss based on the same backpropagated gradients. The increased number of errors in the direction of the gradient means that the image is altered to make the model output classification to be wrong. (Molnar, 2019.) The goal of the attack is to cause the model to misclassify the input image without targeting any specific class (Goodfellow et al., 2014). Figure 19 contains an implementation of the FGSM attack in the Foolbox library. In this research, a default implementation of the attack was used where only changes were made to the epsilon value of the attack.

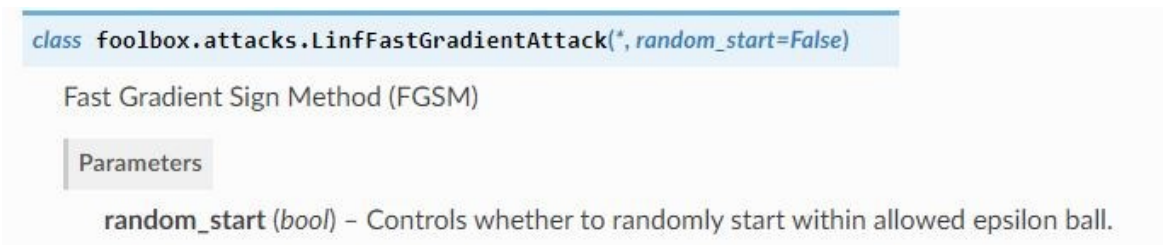


Figure 19. Foolbox implementation of FGSM (Rauber et al., 2017, 2020)

The *Projected Gradient Descent* (PGD) attack created by Madry et al. (2017) is much like the FGSM attack but it works iteratively. Because of this, it has been called in literature as Iterative-Fast Gradient Sign Method. In a PGD attack, the adversarial examples are created by iteratively applying the FGSM attack. PGD starts with randomly initialized perturbation which is then updated at each step. After the first random perturbation, PGD uses the sign of the gradient and updates the pixel values of the input image to the direction of the most considerable loss. This approach tries to find a perturbation that maximizes the loss while having the size of the perturbation small. (Madry et al., 2017.) Figure 20 contains the Foolbox implementation of the PGD attack. In this research, a default implementation of the attack was used where only changes were made to the epsilon value of the attack.

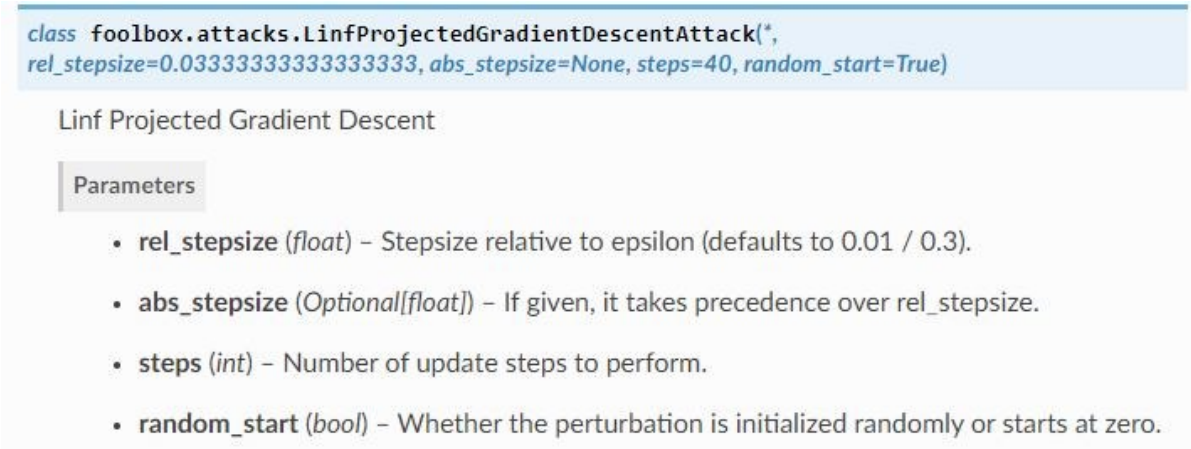


Figure 20. Foolbox implementation of PGD (Rauber et al., 2017, 2020)

6.2 Crafting adversarial examples

Gradient-based attacks in the Foolbox library demand an epsilon value to be configured when crafting an adversarial example. Epsilon value controls the amount of perturbation that is added to the adversarial example. In this research, the aim was to measure the sensitivity of object detection models against the adversarial perturbation by analyzing the obtained AP value when the model is evaluated with the dataset containing perturbation versus the clean dataset. This means datasets should contain different levels of perturbation to see how the models react in each case.

The experiments to select epsilon values for crafting adversarial examples, that would be combined into adversarial datasets, were done by crafting adversarial examples using a combination of different model architectures and gradient-based attacks while changing the epsilon value. The epsilon values used with the FGSM attack varied between 0 and 0.1 with an increase of 0.01 on each step, and with the PGD attack between 0 and 0.15 with an increase of 0.01 on each step. Besides just visualizing the images with perturbation, a PSNR value between the adversarial example and the original image was calculated to have a metric for the level of noise that the adversarial example contains. The PSNR calculation was completed using the tf.image.psnr function from TensorFlow 2 API and by following the example of computing the PSNR over tf.float32 Tensors (Abadi et al., 2015).

The PSNR value between the original sample image and the adversarial example on the FGSM attack with the epsilon value of 0.02 was calculated to be between 29-30, with the epsilon value of 0.05 between 25-26, and with the epsilon value of 0.1 between 20-21, regardless of the model architecture. The PSNR value between the original sample image and the adversarial example on the PGD attack with the epsilon value of 0.02 was calculated to be between 30-31, with the epsilon value of 0.05 between 29-30, and with the epsilon value of 0.1 between 25-26, regardless of the model architecture. Visualization of how the noise increases on different epsilon values when using MobileNet architecture on the FGSM attack can be seen in Figure 21 and on the PGD attack in Figure 22.

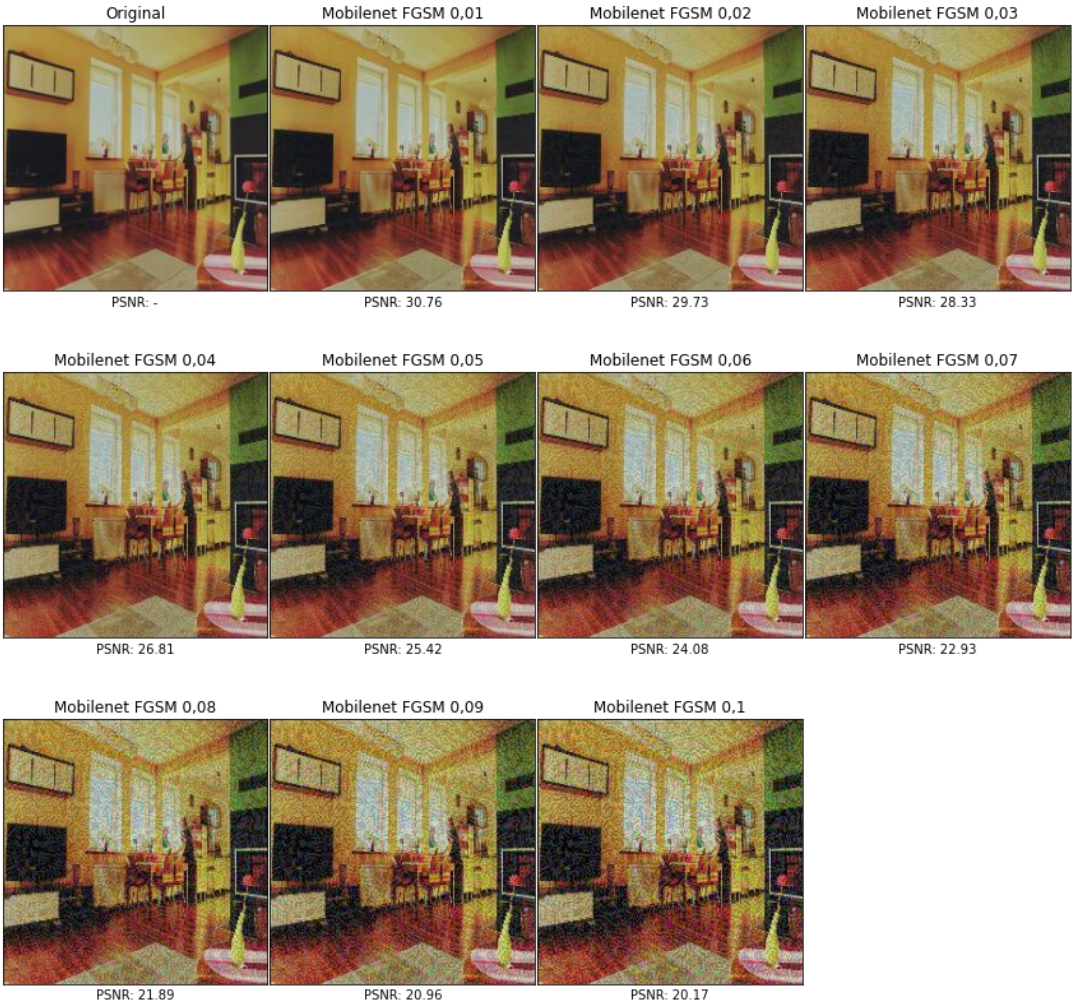


Figure 21. FGSM crafted adversarial examples on image adapted from COCO dataset (Lin et al., 2017)

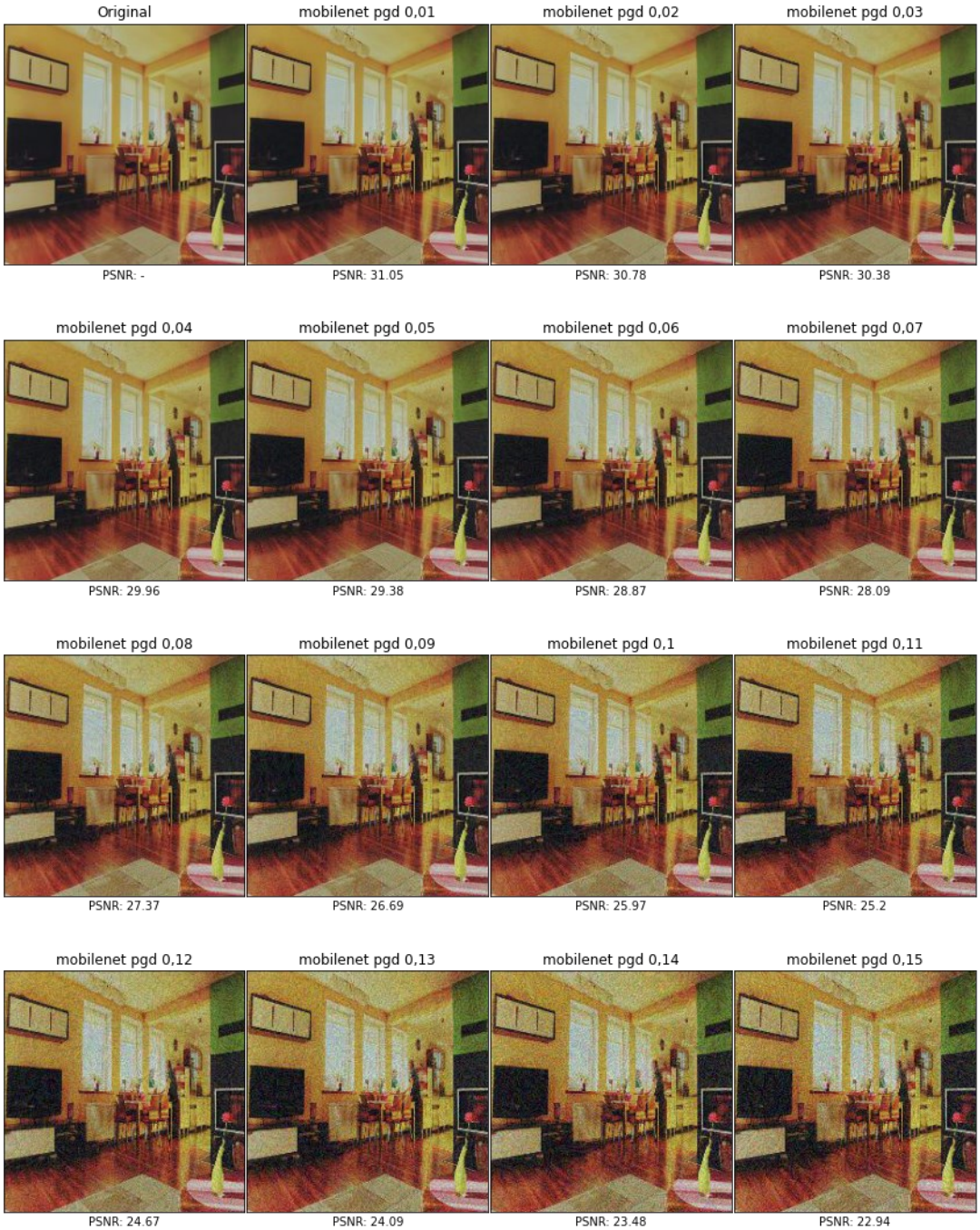


Figure 22. PGD crafted adversarial examples on image adapted from COCO dataset (Lin et al., 2017)

Looking at the two image collages above, it can be seen that with the FGSM attack the noise becomes more visible than with the PGD attack. This was the case regardless of the selected model architecture. To evaluate models with different levels of perturbed noise,

the epsilon values of 0.02, 0.05, and 0.1 were chosen to be used to craft the full adversarial datasets for the evaluation stage. These values would provide the view of how much the amount of perturbation in the image affects to AP metrics of the model while the adversarial example is still recognizable to humans.

After selecting the model, attack algorithm, and epsilon values for the attack, it was noticed that the Foolbox was able to only produce perturbed images in the size of 224x224 as default, due to models and preprocessing steps used in the library. This meant that produced adversarial examples were smaller than the original images on the coco validation dataset. To achieve comparable results, a test was made to separate the perturbation from an adversarial example, then resize the perturbation to the size of the original image, and then apply the perturbation on top of the original image. However, this produced a notably lighter image than what the original was. Another test was done to resize the adversarial example to the size of the original image. However, this made all images too blurry compared to the original image, so the result was not applicable. Results from these tests can be seen in Figure 23.

Figure 23. Results of perturbation resize tests on image adapted from COCO dataset (Lin et al., 2017)

After failing to resize the adversarial example to the size of the original image it was decided that one more dataset would be made where the original images would be resized to the size of 224x224. This would make the AP metric comparisons possible. Figure 24 visualizes resized sample images without perturbation, the amount of perturbation that is added on crafting, and the output image containing the adversarial perturbation.

Figure 24. Demonstrating adversarial perturbation on images adapted from COCO dataset (Lin et al., 2017)

6.3 Creating datasets for evaluation

Adversarial datasets were created by running the coco validation dataset through a pipeline built in jupyter notebook containing Foolbox preprocessing and adversarial example crafting methods. As some of the images on the COCO validation dataset are grayscale these were converted to the RGB images before crafting an adversarial version of them to avoid dimension errors. Crafted adversarial examples were then saved in jpg format. The crafting pipeline creates a copy of a COCO validation dataset but with images resized to 224x224 containing adversarial perturbation. This pipeline was run with all combinations of model architecture, adversarial attack algorithm, and selected epsilon values. The comparable dataset without adversarial perturbation was obtained by running the COCO validation dataset through another jupyter notebook that resized each image into a size of 224x224. A list of datasets used in this research can be seen in Table 2, with names used to refer to them.

Table 2. Datasets used in research

Dataset Name	Base dataset	Image size	Architecture	Attack	Epsilon	Dataset Average PSNR
coco-original	COCO 2017 val	Original	n/a	n/a	n/a	n/a
coco-224	COCO 2017 val	224x224	n/a	n/a	n/a	n/a
MN-0.02-FGSM	COCO 2017 val	224x224	Mobilenet V2	FGSM	0.02	28.2
MN-0.02-PGD	COCO 2017 val	224x224	Mobilenet V2	PGD	0.02	29.1
MN-0.05-FGSM	COCO 2017 val	224x224	Mobilenet V2	FGSM	0.05	24.9
MN-0.05-PGD	COCO 2017 val	224x224	Mobilenet V2	PGD	0.05	27.9
MN-0.1-FGSM	COCO 2017 val	224x224	Mobilenet V2	FGSM	0.1	20.4
MN-0.1-PGD	COCO 2017 val	224x224	Mobilenet V2	PGD	0.1	25.1
RN-0.02-FGSM	COCO 2017 val	224x224	Resnet50	FGSM	0.02	28.1
RN-0.02-PGD	COCO 2017 val	224x224	Resnet50	PGD	0.02	29.1
RN-0.05-FGSM	COCO 2017 val	224x224	Resnet50	FGSM	0.05	24.9
RN-0.05-PGD	COCO 2017 val	224x224	Resnet50	PGD	0.05	27.9
RN-0.1-FGSM	COCO 2017 val	224x224	Resnet50	FGSM	0.1	20.4
RN-0.1-PGD	COCO 2017 val	224x224	Resnet50	PGD	0.1	25.1
EN-0.02-FGSM	COCO 2017 val	224x224	Efficientnet0	FGSM	0.02	28.2
EN-0.02-PGD	COCO 2017 val	224x224	Efficientnet0	PGD	0.02	28.9
EN-0.05-FGSM	COCO 2017 val	224x224	Efficientnet0	FGSM	0.05	25.0
EN-0.05-PGD	COCO 2017 val	224x224	Efficientnet0	PGD	0.05	27.7
EN-0.1-FGSM	COCO 2017 val	224x224	Efficientnet0	FGSM	0.1	20.6
EN-0.1-PGD	COCO 2017 val	224x224	Efficientnet0	PGD	0.1	25.0

The ground truth files for the 224x224 size datasets were created from the coco-original dataset ground truth file, which is available in JSON format on cocodataset.org. The creation of ground truth files was done by using a script written into a jupyter notebook. The script would take a ground truth of an image from the JSON file and convert the bounding boxes to match the image size of 224x224 and then rewrite this into a new JSON file. This approach was validated by using the FiftyOne (Moore & Corso, 2020) tool to visualize the datasets with the ground truths to see that these were correctly shown on images after the resizing. The converted ground truth file was used with all datasets of size 224x224 as the images are the same, with the only difference being the amount of perturbation in them. A view from the FiftyOne tool with the coco-224 dataset and ground truths can be seen in Figure 25.

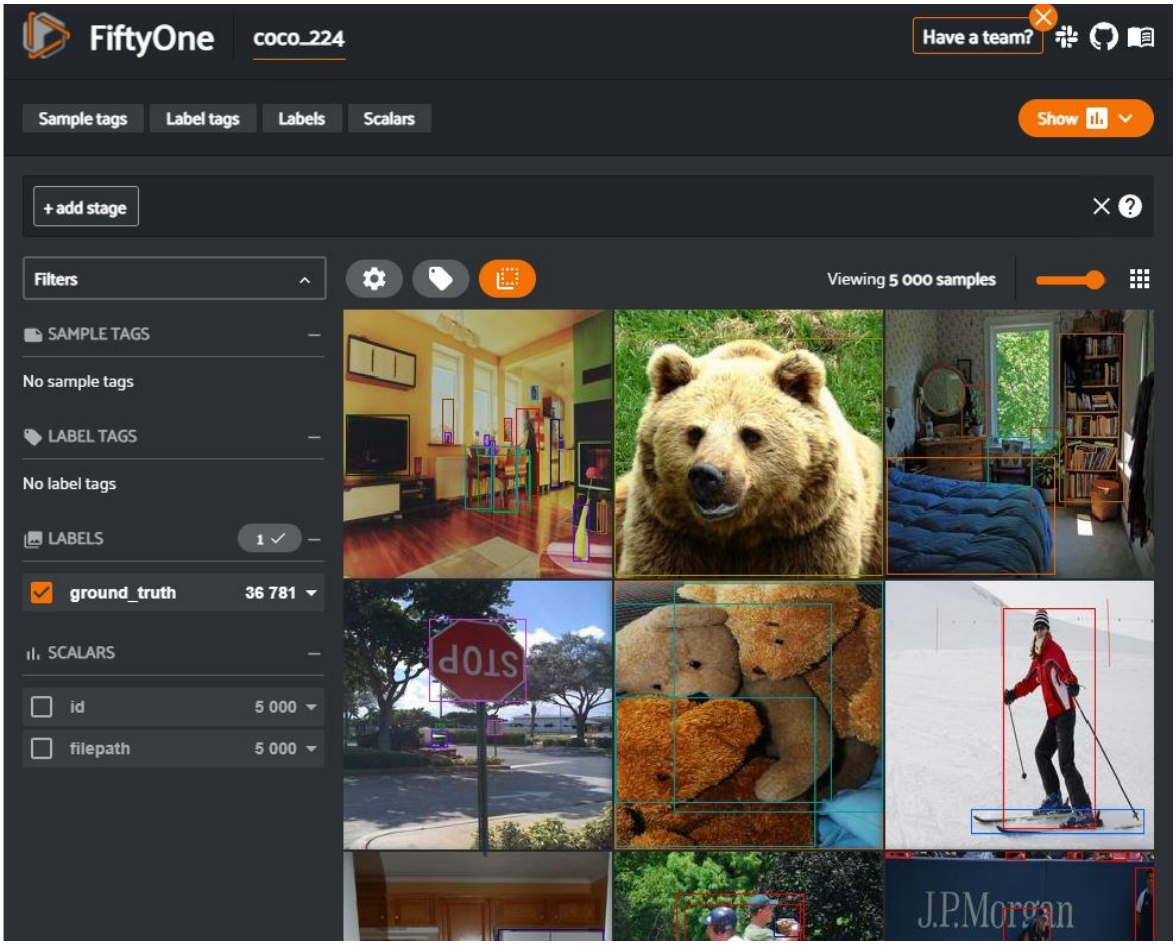


Figure 25. FiftyOne visualization of images and annotations adapted from COCO dataset (Lin et al., 2017)

6.4 Model evaluations and detection visualizations on Google Colab

The evaluation of the pre-trained object detection models was completed using the model_main_tf2.py script from the TensorFlow 2 Object Detection API. TF2 Object Detection API uses TFRecord format to train and evaluate the object detection model and provides a python script create_coco_tf_record.py for transforming the dataset into this format. All datasets used in this research were transformed into the TFRecord format before running the evaluations with the models. The evaluations were run on a Google Colab Notebook using Tensorflow version 2.20 and the installation of TF2 2 Object Detection API. The datasets and corresponding tfrecords were imported from Google Drive to the notebook, while the object detection models were downloaded from Tensorflow 2 Detection Model

Zoo. Google Colaboratory (Colab) offers the possibility to run Colab notebooks on Google's cloud servers while leveraging Google's hardware, including GPUs and TPUs on the browser. This approach was chosen due to insufficient resources on the local laptop. The research was completed with the use of Google's GPUs and on most runs, the NVIDIA Tesla K80 was acquired.

After installing the environment, and uploading the datasets and models into the Colab notebook, the pipeline configuration file was updated to point to the model being evaluated, the tfrecords of the selected dataset, and to mscoco_label_map.pbtxt file, which maps integers of a class to a display name. This file was then used as a configuration file for the model_main_tf2.py script for evaluations. The pipeline in the evaluation notebook was first tested by running the evaluations on the coco-original dataset to confirm that the configuration works. The obtained AP results were compared to the AP results of the model in Table 1 and ensured that the results were the same. The evaluations were then run with the rest of the datasets, in a way that each object detection model in Table 1 was evaluated with each dataset in Table 2. The evaluation results were collected into text files and exported into excel tables to get better visibility of the results. In excel the data was processed to contain only the relevant AP results for this research as can be seen in Table 3. Comprehensive AP results can be found in Appendix A.

Table 3. Model AP results in excel table

Epsilon value 0.05	CenterNet Resnet101 V1 FPN 512x512	CenterNet Resnet50 V1 FPN 512x512	CenterNet Resnet50 V2 512x512	EfficientDet D0 512x512	SSD MobileNet v2 320x320	SSD MobileNet V2 FPNLite 320x320	SSD MobileNet V2 FPNLite 640x640	
Speed (ms)	34,0	27,0	27,0	39,0	19,0	22,0	39,0	
Tensorflow COCO (mAP)	34,2	31,2	29,5	33,6	20,2	22,2	28,2	
coco-original	34,2	31,2	29,5	33,6	20,2	22,2	28,1	
coco-224	24,1	20,5	19,8	26,9	18,1	18,7	19,2	**PSNR avg.**
MN-0.05-FGSM	13,8	10,6	10,9	17,2	11,1	10,6	9,2	24,88
MN-0.05-PGD	20,5	16,8	16,6	24,0	16,3	15,9	15,3	27,89
RN-0.05-FGSM	13,8	11,0	11,0	17,6	11,6	11,2	9,4	24,85
RN-0.05-PGD	20,6	16,9	16,5	24,2	16,3	16,0	15,4	27,87
EN-0.05-FGSM	13,3	10,4	10,7	16,4	10,7	10,3	9,0	25,03
EN-0.05-PGD	18,9	15,6	15,3	22,6	15,0	14,6	13,8	27,69

Another Google Colab notebook was created for visualizing the detections. This notebook uses the TF2 Object Detection API to interact with the selected model and produces the detection results, which are then visualized by drawing a bounding box around the object with detection class and detection score. When used together with the AP results, visuali-

zation provides a supportive aspect for understanding the effects that adversarial perturbation causes on the detections. The visualization notebook was tested with a small set of the same images from the different datasets in Table 2. Figure 26 shows an example visualization of the detections made by the EffientDet D0 object detection model when provided a sample image from coco-224 and EN-based datasets. Detections from other models with the same sample image on different attack parameters can be found in Appendix B.

Figure 26. Visualization of detections made by EfficientDet D0 512x512 model on image adapted from COCO dataset (Lin et al., 2017)

7 RESULTS

This chapter presents the results obtained from the evaluation notebook in which the evaluations were run on the models in Table 1 with datasets in Table 2 using the model_main_tf2.py script. While presenting and inspecting the results from the experiment, the main focus was to find out what kind of effect the adversarial perturbation included in images has on detection outcomes when observing the AP metric. To answer the research question "how do the lightweight object detection models react to adversarial examples" the following leading questions are used:

- How does the attack algorithm affect the detection results?
- Does the model architecture in adversarial crafting affect the detection results?
- How much effect does the noise in the image have on PSNR and AP results?
- Are some researched models more robust against adversarial examples than others?

The first subchapter presents the AP results after resizing the images and compares the coco-224 dataset to the coco-original. The second subchapter presents the results of evaluations when datasets were created using the FGSM attack algorithm on different epsilon values. The adversarial datasets created with the FGSM attack are compared to the coco-224 dataset. In the third subchapter, results from evaluations with datasets created using the PGD attack algorithm on different epsilon values are presented. The adversarial datasets created with the PGD attack are compared to the coco-224 dataset. Figure 27 shows the obtained AP results in a graph while each subchapter includes a table of the related results.

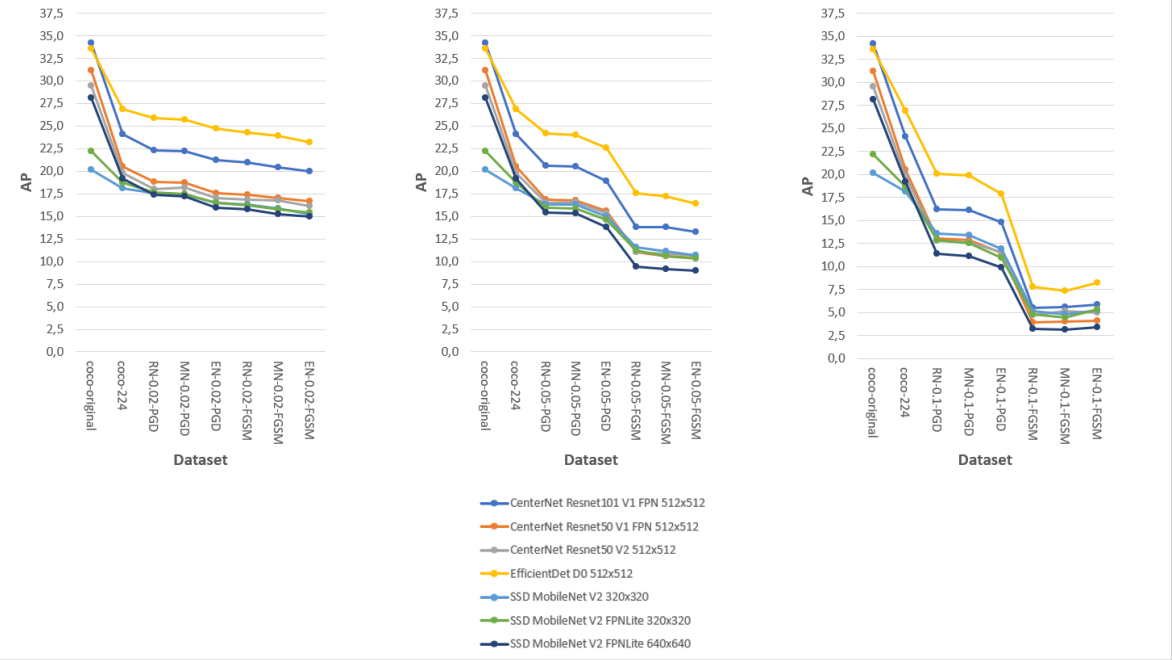


Figure 27. Excel graph of models AP results with datasets

7.1 Results on resized images

This chapter briefly shows the AP results when models were evaluated with the coco-224 dataset and compared to the coco-original dataset. The drop percentage in AP is rounded to the nearest one while AP is rounded to the nearest 10^{th} to make the results more readable.

The CenterNet models in this research are trained with images scaled to 512x512. After running the evaluations and comparing the results, the following observations were made: CenterNet Resnet101 model AP drops 30% from 34.2 to 24.1, while CenterNet Resnet50 V1 model AP drops 34% from 31.2 to 20.5, and the CenterNet Resnet50 V2 model AP drops 33% from 29.5 to 19.8.

The EfficientDet D0 model is also trained with images scaled to 512x512. After running the evaluations and comparing the results it was observed that the AP drops 20% from 33.6 to 26.9.

The SSD Mobilenet V2 320x320 and SSD Mobilenet V2 FPNLite 320x320 are trained with images scaled to 320x320 while the SSD MobileNet V2 FPNLite 640x640 model is

trained with images scaled to 640x640. After running the evaluations and comparing the results the following observations were made: AP of the 320x320 model without the FPN-Lite feature extractor drops 10% from 20.2 to 18.1, while with the FPNLite the drop is 16% from 22.2 to 18.7. AP value of the SSD MobileNet V2 FPNLite 640x640 model drops 32% from 28.1 to 19.2.

From the results in Table 4, it can be noticed that the AP value drops more with the models that have been trained with bigger image sizes and then evaluated with the coco-224 dataset. From here on the AP results of model evaluations with adversarial datasets are compared to the results obtained with coco-224 instead of coco-original. This solution invalidates the difference coming from the image resizing step as images in the coco-224 dataset and adversarial datasets are the same size.

Table 4. Model AP values on the coco-224 dataset

Model	coco-original	Drop	coco-224
CenterNet Resnet101 V1 FPN 512x512	34,2	30 %	24,1
CenterNet Resnet50 V1 FPN 512x512	31,2	34 %	20,5
CenterNet Resnet50 V2 512x512	29,5	33 %	19,8
EfficientDet D0 512x512	33,6	20 %	26,9
SSD MobileNet V2 320x320	20,2	10 %	18,1
SSD MobileNet V2 FPNLite 320x320	22,2	16 %	18,7
SSD MobileNet V2 FPNLite 640x640	28,1	32 %	19,2

7.2 Results with FGSM attack

This chapter shows the AP results when models were evaluated with adversarial datasets crafted using the FGSM attack algorithm. The drop percentage in AP is rounded to the nearest one while AP is rounded to the nearest 10^{th} to make the results more readable. Each subchapter contains a result table where the lowest and highest drop percent are highlighted.

7.2.1 FGSM on epsilon value of 0.02

When evaluating the models with the MN-0.02-FGSM dataset, the AP was reduced by an average of 15%. From the evaluated models, the SSD MobileNet V2 FPNLite 640x640 had AP decreased the most as it dropped 21% to 15.2. The least decrease happened with the EfficientDet D0 512x512 model which AP dropped 11% to 24.0.

When evaluating the models with the RN-0.02-FGSM datasets, the AP was reduced by an average of 13%. From the evaluated models, the SSD MobileNet V2 FPNLite 640x640 had AP decreased the most as it dropped 18% to 15.8. The least decrease happened with the EfficientDet D0 512x512 model which AP dropped 10% to 24.3.

When evaluating the models with the EN-0.02-FGSM dataset, the AP was reduced by an average of 18%. From the evaluated models, the SSD MobileNet V2 FPNLite 640x640 had AP decreased the most as it dropped 22% to 15.0. The least decrease happened with the EfficientDet D0 512x512 model which AP dropped 14% to 23.2.

From the results in Table 5, it can be noticed that the AP metric drops most when using the EN-0.02-FGSM dataset and least when using the RN-0.02-FGSM dataset. In overall, the drop in AP with this epsilon value and attack is between 10 to 22 percent, depending on the evaluated model. From the evaluated models, it seems that the EfficientDet D0 512x512 model is most robust against the adversarial perturbation crafted with FGSM attack on epsilon value of 0.02, while the MobileNet V2 FPNLite 640x640 model performs worst.

Table 5. Model AP values on FGSM-0.02 datasets

Model	coco-224	Drop	MN-0.02-FGSM	Drop	RN-0.02-FGSM	Drop	EN-0.02-FGSM
CenterNet Resnet101 V1 FPN 512x512	24,1	15 %	20,5	13 %	21,0	17 %	20,0
CenterNet Resnet50 V1 FPN 512x512	20,5	17 %	17,1	15 %	17,4	19 %	16,7
CenterNet Resnet50 V2 512x512	19,8	15 %	16,7	15 %	16,8	19 %	16,1
EfficientDet D0 512x512	26,9	11 %	24,0	10 %	24,3	14 %	23,2
SSD MobileNet V2 320x320	18,1	12 %	15,9	10 %	16,3	16 %	15,2
SSD MobileNet V2 FPNLite 320x320	18,7	16 %	15,8	13 %	16,2	18 %	15,4
SSD MobileNet V2 FPNLite 640x640	19,2	21 %	15,2	18 %	15,8	22 %	15,0

7.2.2 FGSM on epsilon value of 0.05

When evaluating the models with the MN-0.05-FGSM dataset the AP was reduced by an average of 44%, with the RN-0.05-FGSM dataset by an average of 42%, and with the EN-0.05-FGSM dataset by an average of 45%. From the results in Table 6, it can be noticed that the AP metric drops most when using the EN-0.05-FGSM dataset and least when using the RN-0.05-FGSM dataset. In overall, the drop in AP with this epsilon value and attack is between 35 to 53 percent, depending on the evaluated model. From the evaluated models, it seems that the EfficientDet D0 512x512 model is most robust against the adversarial perturbation crafted with FGSM attack on epsilon value of 0.05, while the MobileNet V2 FPNLite 640x640 model performs worst.

Table 6. Model AP values on FGSM-0.05 datasets

Model	coco-224	Drop	MN-0.05-FGSM	Drop	RN-0.05-FGSM	Drop	EN-0.05-FGSM
CenterNet Resnet101 V1 FPN 512x512	24,1	43 %	13,8	43 %	13,8	45 %	13,3
CenterNet Resnet50 V1 FPN 512x512	20,5	48 %	10,6	46 %	11,0	49 %	10,4
CenterNet Resnet50 V2 512x512	19,8	45 %	10,9	44 %	11,0	46 %	10,7
EfficientDet D0 512x512	26,9	36 %	17,2	35 %	17,6	39 %	16,4
SSD MobileNet V2 320x320	18,1	39 %	11,1	36 %	11,6	41 %	10,7
SSD MobileNet V2 FPNLite 320x320	18,7	43 %	10,6	40 %	11,2	45 %	10,3
SSD MobileNet V2 FPNLite 640x640	19,2	52 %	9,2	51 %	9,4	53 %	9,0

7.2.3 FGSM on epsilon value of 0.1

When evaluating models with the MN-0.1-FGSM dataset the AP was reduced by an average of 77%, with the RN-0.1-FGSM dataset by an average of 76%, and with the EN-0.1-FGSM dataset by an average of 75%. From the results in Table 7, it can be noticed that the AP metric drops most when using the MN-0.1-FGSM dataset and least when using the EN-0.1-FGSM dataset. In overall, the drop in AP with this epsilon value and attack is between 70 to 84 percent, depending on the evaluated model. The high drop in AP value is understandable with these parameters as images in datasets are containing a lot of noise. From the evaluated models, it seems that the EfficientDet D0 512x512 model is most robust against the adversarial perturbation crafted with FGSM attack on epsilon value of 0.1, while the MobileNet V2 FPNLite 640x640 model performs worst. However, it should be noticed that all AP results are below 10 which means that models are not detecting objects

correctly in images. This means that models are pretty unsuitable for any detection tasks at this noise level.

Table 7. Model AP values on FGSM-0.1 datasets

Model	coco-224	Drop	MN-0.1-FGSM	Drop	RN-0.1-FGSM	Drop	EN-0.1-FGSM
CenterNet Resnet101 V1 FPN 512x512	24,1	77 %	5,6	77 %	5,5	76 %	5,9
CenterNet Resnet50 V1 FPN 512x512	20,5	80 %	4,0	81 %	4,0	80 %	4,2
CenterNet Resnet50 V2 512x512	19,8	74 %	5,1	76 %	4,8	75 %	5,0
EfficientDet D0 512x512	26,9	73 %	7,4	71 %	7,8	70 %	8,2
SSD MobileNet V2 320x320	18,1	73 %	4,8	72 %	5,1	71 %	5,2
SSD MobileNet V2 FPNLite 320x320	18,7	76 %	4,5	74 %	4,8	72 %	5,3
SSD MobileNet V2 FPNLite 640x640	19,2	84 %	3,1	83 %	3,2	82 %	3,4

7.3 Results with PGD attack

This chapter shows the AP results when models were evaluated with adversarial datasets crafted using the PGD attack algorithm. The drop percentage in AP is rounded to the nearest one while AP is rounded to the nearest 10^{th} to make the results more readable. Each subchapter contains a result table where the lowest and highest drop percent are highlighted.

7.3.1 PGD on epsilon value of 0.02

When evaluating the models with the MN-0.02-PGD dataset, the AP was reduced by an average of 7%. From the evaluated models, the SSD MobileNet V2 FPNLite 640x640 model had AP decreased the most as it dropped 10% to 17.3. The least decrease happened with the SSD MobileNet V2 320x320 model where AP dropped only 4% to 17.4.

When evaluating the models with the RN-0.02-PGD dataset, the AP was reduced by an average of 7%. From the evaluated models, the SSD MobileNet V2 FPNLite 640x640 model had AP decreased the most as it dropped 9% to 17.4. The least decrease happened with the SSD MobileNet V2 320x320 model where AP dropped only 3% to 17.5.

When evaluating the models with the EN-0.02-PGD dataset, the AP was reduced by an average of 12%. From the evaluated models, the SSD MobileNet V2 FPNLite 640x640

model had AP decreased the most as it dropped 17% to 16.0. The least decrease happened with the EfficientDet D0 512x512 model where AP dropped 8% to 24.7.

From the results in Table 8, it can be noticed that the AP metric drops most when using the EN-0.02-PGD dataset, and least when using the RN-0.02-PGD dataset, even though the results between this and the MN-0.02-PGD dataset are very similar. In overall, the drop with this epsilon value and attack is between 3 to 17 percent depending on the evaluated model. The small drop in AP is understandable because the noise level in the images is not as high as it is with the datasets created with the FGSM attack. From the evaluated models, it seems that the SSD MobileNet V2 320x320 model and the EfficientDet D0 512x512 model are most robust against the adversarial perturbation crafted with PGD attack on epsilon value of 0.02 if the drop percentage is to be considered. However the latter achieves better AP results in general. Of all reviewed models the MobileNet V2 FPNLite 640x640 performs worst.

Table 8. Model AP values on PGD-0.02 datasets

Model	coco-224	Drop	MN-0.02-PGD	Drop	RN-0.02-PGD	Drop	EN-0.02-PGD
CenterNet Resnet101 V1 FPN 512x512	24,1	8 %	22,2	7 %	22,3	12 %	21,2
CenterNet Resnet50 V1 FPN 512x512	20,5	9 %	18,7	8 %	18,8	14 %	17,6
CenterNet Resnet50 V2 512x512	19,8	8 %	18,2	9 %	18,0	14 %	17,1
EfficientDet D0 512x512	26,9	4 %	25,7	4 %	25,9	8 %	24,7
SSD MobileNet V2 320x320	18,1	4 %	17,4	3 %	17,5	9 %	16,5
SSD MobileNet V2 FPNLite 320x320	18,7	6 %	17,5	5 %	17,7	12 %	16,5
SSD MobileNet V2 FPNLite 640x640	19,2	10 %	17,3	9 %	17,4	17 %	16,0

7.3.2 PGD on epsilon value of 0.05

When evaluating the models with the MN-0.05-PGD dataset the AP was reduced by an average of 15%, with the RN-0.05-PGD dataset also by an average of 15%, and with the EN-0.05-PGD dataset by an average of 22%. From the results in Table 9, it can be noticed that the AP metric drops most when using the EN-0.05-PGD dataset, and least when using the RN-0.05-PGD dataset, even though the results between this and the MN-0.05-PGD dataset are very similar. In overall, the drop in AP with this epsilon value and attack is between 10 to 28 percent depending on the evaluated model. From the evaluated models, it seems that the SSD MobileNet V2 320x320 model and the EfficientDet D0 512x512 mod-

el are most robust against the adversarial perturbation crafted with PGD attack on epsilon value of 0.05 if the drop percentage is to be considered. However the latter achieves better AP results in general. Of all reviewed models the MobileNet V2 FPNLite 640x640 performs worst.

Table 9. Model AP values on PGD-0.05 datasets

Model	coco-224	Drop	MN-0.05-PGD	Drop	RN-0.05-PGD	Drop	EN-0.05-PGD
CenterNet Resnet101 V1 FPN 512x512	24,1	15 %	20,5	15 %	20,6	22 %	18,9
CenterNet Resnet50 V1 FPN 512x512	20,5	18 %	16,8	18 %	16,9	24 %	15,6
CenterNet Resnet50 V2 512x512	19,8	16 %	16,6	17 %	16,5	23 %	15,3
EfficientDet D0 512x512	26,9	11 %	24,0	10 %	24,2	16 %	22,6
SSD MobileNet V2 320x320	18,1	10 %	16,3	10 %	16,3	17 %	15,0
SSD MobileNet V2 FPNLite 320x320	18,7	15 %	15,9	14 %	16,0	22 %	14,6
SSD MobileNet V2 FPNLite 640x640	19,2	20 %	15,3	20 %	15,4	28 %	13,8

7.3.3 PGD on epsilon value of 0.1

When evaluating models with the MN-0.1-PGD dataset the AP was reduced by an average of 33%, with the RN-0.1-PGD dataset by an average of 32%, and with the EN-0.1-PGD dataset by an average of 40%. From the results in Table 10, it can be noticed that the AP metric drops most when using the EN-0.1-PGD dataset and least when using the RN-0.1-PGD dataset. In overall, the drop with this epsilon value and attack is between 25 to 49 percent depending on the evaluated model. From the evaluated models, it seems that the SSD MobileNet V2 320x320 model and EfficientDet D0 512x512 model are most robust against the adversarial perturbation crafted with PGD attack on epsilon value of 0.1 if the drop percentage is to be considered. However the latter achieves better AP results in general. Of all reviewed models the MobileNet V2 FPNLite 640x640 performs worst.

Table 10. Model AP values on PGD-0.1 datasets

Model	coco-224	Drop	MN-0.1-PGD	Drop	RN-0.1-PGD	Drop	EN-0.1-PGD
CenterNet Resnet101 V1 FPN 512x512	24,1	33 %	16,2	33 %	16,2	39 %	14,8
CenterNet Resnet50 V1 FPN 512x512	20,5	37 %	12,8	36 %	13,0	44 %	11,5
CenterNet Resnet50 V2 512x512	19,8	36 %	12,6	36 %	12,8	42 %	11,5
EfficientDet D0 512x512	26,9	26 %	19,9	25 %	20,1	33 %	17,9
SSD MobileNet V2 320x320	18,1	26 %	13,4	25 %	13,6	34 %	11,9
SSD MobileNet V2 FPNLite 320x320	18,7	33 %	12,5	31 %	12,8	41 %	10,9
SSD MobileNet V2 FPNLite 640x640	19,2	42 %	11,2	41 %	11,4	49 %	9,9

8 DISCUSSION

This chapter presents observations from acquired results, discusses the limitations of the research, and gives ideas for future research work. In the first subchapter, the observations made from the evaluation results are presented. The second subchapter covers observations made from the PSNR value of the adversarial examples. In the third subchapter, the robustness of lightweight object detection models used in the research is discussed. The fourth subchapter discusses the limitations of the research and mentions some future research ideas.

8.1 Observations based on evaluation results

The first noticeable observation can be made from the results after the image resizing method, which was made to obtain comparable results between clean and adversarially perturbed images. The resizing step has reduced the AP metric of all reviewed models, and with the models trained on larger image sizes, the drop is even higher than with others. However, as this step was made to obtain comparable results from experiments, I am not going to cover this topic more at this point as the focus will be on results obtained with adversarial datasets.

When comparing the selected gradient-based attacking algorithms FGSM and PGD used in this research, the results show that with the FGSM, the models' AP is reduced more than with the PGD when the same epsilon value is used. Another note that can be made is that when the epsilon value gets higher, the evaluated AP results are getting lower with both attack methods, which is understandable as images on the dataset are containing more noise. Looking at the visualization and PSNR value of crafted adversarial examples, it can be noticed that the amount of noise added to the images on Foolbox is higher with the FGSM method in Figure 21 compared to the PGD method in Figure 22. While using the same epsilon, the PSNR value of images crafted with FGSM is noticeably lower than with PGD. Reflecting on these results, it could be argued that the adversarial examples crafted with FGSM methods contain more noise than with PGD, at least if the same epsilon value

is used. When focusing only on the epsilon value used to craft adversarial examples on Foolbox, the results show that with a higher value a more perturbation is added to the image. This can be verified by looking at the PSNR value of the image, visualizing the image with adversarial perturbation, and looking at the AP result of models evaluated with adversarial datasets. The higher the epsilon value is when creating the adversarial dataset, the noisier are images included in the dataset, and the lower AP metrics are obtained.

From the evaluation results following observations were made regarding the model architecture in crafting adversarial examples. The adversarial datasets crafted with the ResNet50 architecture seemed to be least effective in reducing the model AP value while the datasets crafted with the EfficientNetB0 architecture reduced it the most in the majority of cases. It was also noticed, that the dispersion between the AP results and attack algorithm was slightly higher when using the PGD attack instead of FGSM. When analyzing the architecture used to craft adversarial datasets in Foolbox and the base architecture used by the object detection model, using the same architecture did not produce lower AP results compared to the case where these architectures were different. This can be seen in results with models using the MobilenetV2 base architecture that were evaluated with MN-datasets, or with models using the ResNet50 base architecture that were evaluated with RN-datasets.

8.2 Observations based on PSNR value

When looking at the average PSNR value of the datasets in Table 2, the average PSNR value for crafted datasets gets lower when the epsilon value is higher, meaning that the amount of adversarial perturbation in images can be recognized as a level of noise it contains. However, looking at the average PSNR value more closely it can be noticed that this does not always mean lower AP results. The average PSNR value of the EN-0.05-FGSM dataset is 25.0 which is higher than the MN-0.05-FGSM dataset (24.9) or RN-0.05-FGSM dataset (24.9), but the obtained AP results (Table 6) are lower. Similarly, when comparing the AP results of the EN-0.05-FGSM dataset and EN-0.1-PGD dataset, the rounded PSNR value is the same (25.0) but the AP results are worse with the FGSM method. Average PSNR values of the datasets are shown in Appendix A.

According to observations, it can be assumed that if the PSNR value of the image is low, the image contains a high amount of noise making the detection of objects from the image harder, which is then reflected as a low AP results obtained by evaluation. This is understandable because the PSNR is used to provide a measure for human visual perception when comparing the quality of two images, and AP measures the accuracy of the object detection model to make correct predictions. To analyze the suitability of the PSNR metric when measuring the noise level in adversarial examples, the following example can be used: If the PSNR value obtained from an image is high, then this image contains only a small amount of perturbation making it slightly different from the original image. A low PSNR value would then mean that the image contains a high amount of perturbation, making it very different from the original image.

To address the results where the AP metric is worse on one dataset over the other, even if the average PSNR value is higher a few explanations can be made: The average PSNR value does not take a stand on how many objects one image would contain or that the PSNR value between images in the same dataset would be the same. This means that there will be variations in the results. Also when visualizing the crafted adversarial examples with the FGSM attack algorithm, these seemed to have more perturbation than images crafted with the PGD attack algorithm. Another thing that should be acknowledged is that the attacks used to craft adversarial examples were created against image classification models. When evaluating results with object detection models, there could be variations in cases where the amount of adversarial perturbation is more or less the same between datasets because the attack is not optimized for the object detection task. However, in most cases the datasets crafted with the FGSM attack algorithm have lower PSNR values, the images contain more perturbation, and the evaluation provides lower AP results compared to datasets crafted with the PGD attack algorithm.

8.3 Robustness of lightweight object detection models

From the experiments made in this research, the following observations were made regarding the robustness of lightweight object detection models from Tensorflow 2 Detection Model Zoo against adversarial examples. While looking at the AP results in general, the

assumption about the models' vulnerability to adversarial perturbation could be verified as each model had its AP results dropped when evaluated with datasets containing adversarial perturbation. This was the case even though the crafting method used to craft adversarial examples in Foolbox was designed to be used with image classification tasks and it is not optimized to fool object detection models. When visualizing the detections that the models made on adversarially perturbed images in this research, it was noticed that the perturbation added to the images was able to affect the detection results in the following ways: the perturbation vanished the objects from being detected, the detected object was misclassified, or the image was fabricated with additional detections, which caused the model to make false detections. Figure 28 visualizes some of the detections from models with different architectures on images from crafted datasets.

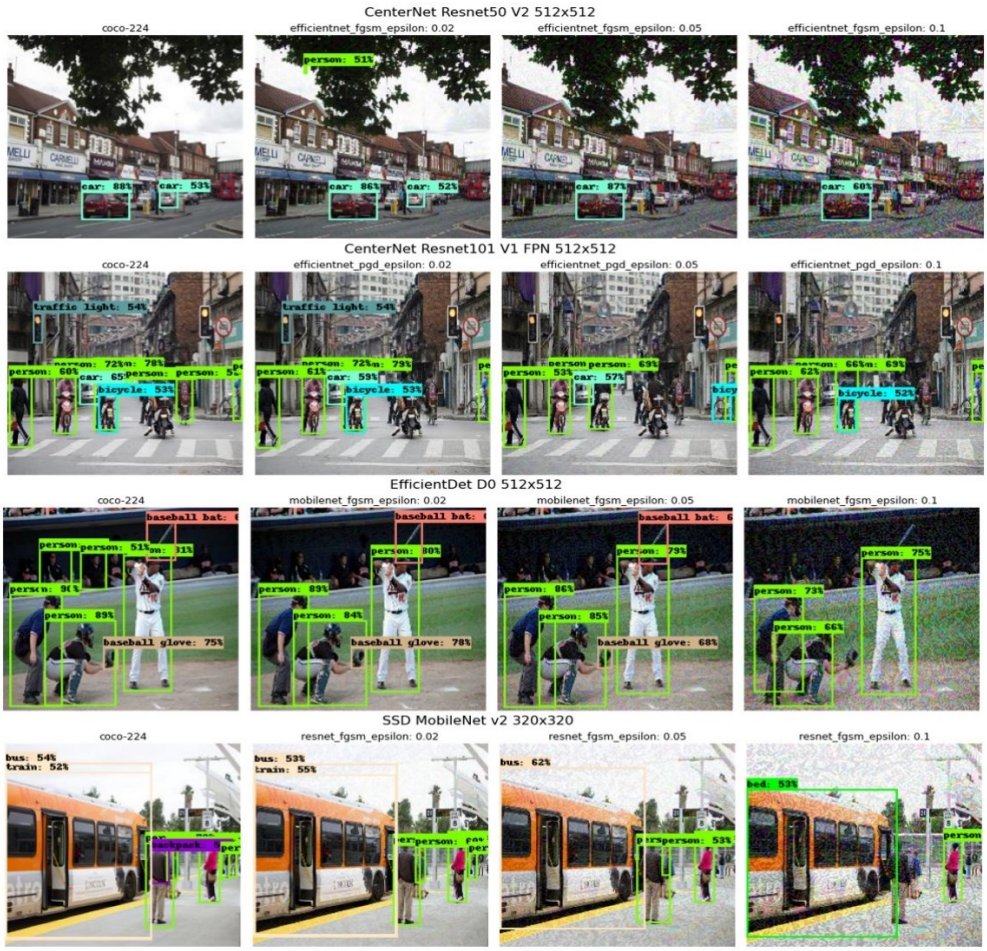


Figure 28. Detection examples from models with different architectures on images adapted from COCO dataset (Lin et al., 2017)

The EfficientDetD0 512x512 object detection model seemed to perform best against the adversarial datasets crafted using the Foolbox implementation of gradient attack methods FGSM and PGD. It achieved the highest AP results against each adversarial dataset, and when evaluated with the datasets crafted using the FGSM attack algorithm, the percentual drop in the AP metric was the lowest of all models. When evaluated with the datasets crafted using the PGD attack algorithm, the percentual drop in the AP metric was almost the same as with the SSD MobileNet V2 320x320 model, as both are having the lowest drop of all evaluated models. However, the overall AP results with the EfficientDetD0 512x512 were notably higher. It can be argued that because the EfficientDetD0 512x512 model was one of the best performing models according to AP results on the coco-original dataset, it would be one of the best with adversarial datasets as well. However, because it performed better than CenterNet Resnet or SSD MobileNet models used in this research, it could also reflect that the model architecture makes it more resistant to adversarial perturbations than the others, but this should be further researched.

The worst performing model in the experiments according to AP metrics was the SSD MobileNet V2 FPNLite 640x640 model. With each adversarial dataset crafted for the experiments, it recorded the lowest AP results while the AP drop percent was also the highest in each case. One reason why this model performed worst could be the fact that it has been trained with images scaled to 640x640 and was now evaluated with images scaled to a size of 224x224, but this would need further research before making any conclusions. Even though it recorded the worst AP results with adversarial datasets, it seemed to resist adversarial perturbations quite well on some sample images as seen in Figure 29.

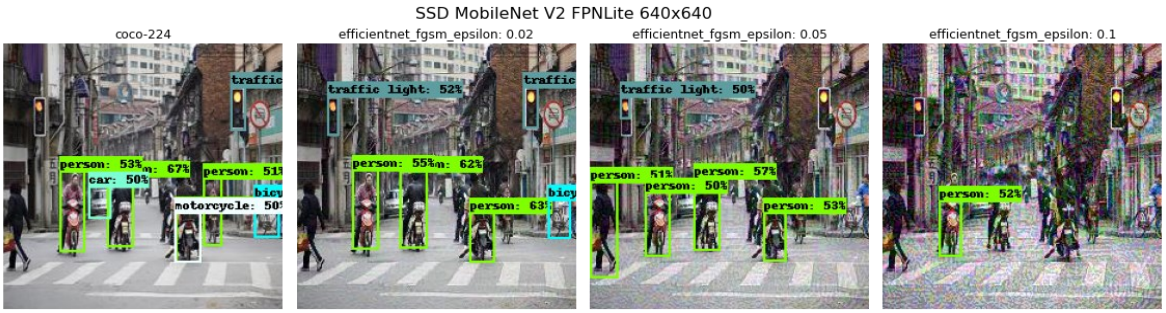


Figure 29. Detection examples from SSD MobileNet V2 FPNLite 640x640 model on image adapted from COCO dataset (Lin et al., 2017)

When analyzing the models' robustness against the architectures used in crafting adversarial datasets, the datasets that were crafted using the EfficientNetB0 architecture seemed to have the best success in decreasing the model accuracy in every other case except with the FGSM attack algorithm and epsilon value of 0.1. In general, the attack algorithm, epsilon value, and the model architecture to craft adversarial examples all affected the accuracy of the model when evaluated with the adversarial datasets. Using a higher epsilon value helps to visualize the perturbation that attacks add to the image as can be seen in Figure 30. This figure shows the difference in perturbations between the architectures, but it should be acknowledged that the perturbation is different in each image of the dataset.

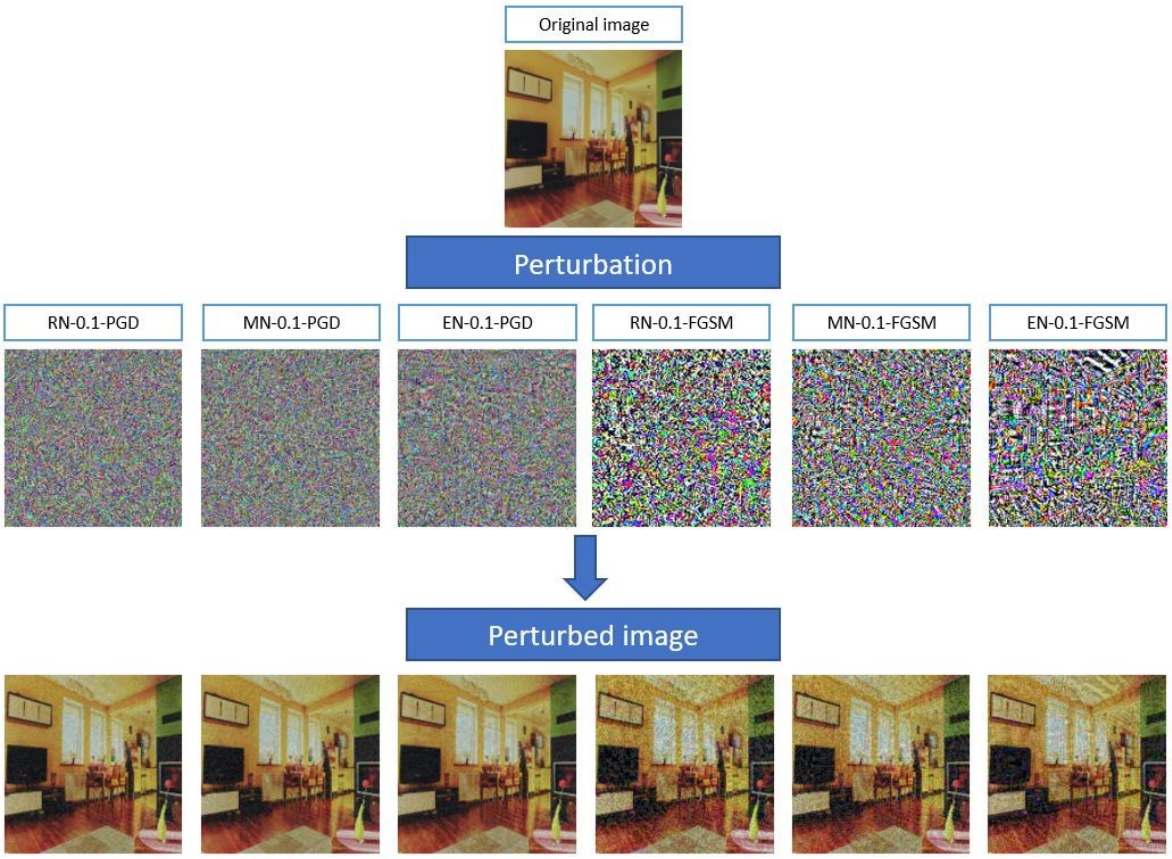


Figure 30. Adversarial perturbation of different architectures on image adapted from COCO dataset (Lin et al., 2017)

The following observations were made regarding the calculated drop percent. The model AP results dropped from results obtained with the coco-224 dataset when evaluated with datasets crafted on epsilon value of 0.02 and FGSM attack between 10–22%, and with

PGD attack between 4–10%. Using an epsilon value of 0.05, the drop with datasets crafted using the FGSM attack was between 35–53%, and with the PGD attack between 10–28%. With an epsilon value of 0.1, the drop on adversarial datasets using FGSM attack was between 70–84%, and with adversarial datasets using PGD attack between 25–49%. On the left side in Figure 31, the object detection models are listed according to their robustness against adversarial perturbation. On the right side, the percentual drop in AP results between the coco-224 dataset and adversarial datasets is presented.

Models arranged by the robustness against adversarial perturbation

1. EfficientDet D0 512x512
2. CenterNet Resnet101 V1 FPN 512x512
3. CenterNet Resnet50 V1 FPN 512x512
4. CenterNet Resnet50 V2 512x512
5. SSD MobileNet v2 320x320
6. SSD MobileNet V2 FPNLite 320x320
7. SSD MobileNet V2 FPNLite 640x640

Percentual drop in AP with different parameters from coco-224

EPSILON	FGSM	PGD
0.02	10 – 22 %	4 – 10 %
0.05	35 – 53 %	10 – 28 %
0.1	70 – 84 %	25 – 49 %

Figure 31. Best performing models and average percentual drop by the attack

Another observation regarding the models used in experiments that could be discussed is that if the model is trained with smaller images, the drop in AP metric is not as significant as with other models. This observation was made when comparing the MobileNet V2 models trained on an image size of 320x320 to the MobileNet V2 model trained on an image size of 640x640. The reason for this could lie in the fact that the images used in this research are 224x224 in size, which is closer to the size of images used to train the 320x320 models, but this should be further investigated.

8.4 Limitations of the research and future work

While the results show that the pre-trained lightweight object detection models are vulnerable to adversarial perturbation, it is hard to make universal conclusions about the sensitivity of the models. Models used in the research were selected from a Tensorflow 2 Detection Model Zoo, which contains only a limited amount of pre-trained models for object

detection, making the scope of research quite narrow. The repository is missing some of the latest lightweight models mentioned in the literature and contains only some of the models mentioned in chapter 3.4. Models from the YOLO family are currently not included in the TF2 Detection Model Zoo, but are under development and could make their way into the repository in the future. However, all the models in Model Zoo are pre-trained on COCO 2017 dataset which makes them a good starting point to be used in transfer learning for a new detection task or used for out-of-the-box interference with existing categories.

What comes to the limitations of crafting the adversarial examples, the Foolbox python library does not contain straight attacks against the Tensorflow 2 Object Detection Models. When searching the python library to craft adversarial examples using different attack algorithms, I did not come across one that could be used specifically for the object detection task or that could use the models straight from Tensorflow 2 Detection Model Zoo. The adversarial examples crafted on Foolbox are designed to fool a Keras image classification model, and therefore they should not be as effective against the object detection models as they would be against image classification models. This was also acknowledged by Wang, Wang, et al. (2020). However, the Keras image classification models selected for crafting the adversarial examples contain the same base architecture as object detection models used in experiments. It was studied if there would be any correlation between these when evaluating the sensitivity of the object detectors, but none could be found. The adversarial perturbation applied to the image is designed to cause the image classifier to misclassify the image. At first, the model in Foolbox predicts the most likely class to the whole image and then crafts a perturbation that should cause the prediction to be misclassified into something else. The process did not contain a check if the image prediction is correct at first or if the prediction is wrong after. Another note that should be made is that the majority of images in the COCO dataset contain objects from multiple classes and selecting only one class from the image might be difficult. However, the crafted adversarial examples in this research should be viewed as images containing adversarially crafted perturbation, which can be simplified to images containing noise as visualizations show.

When thinking about limitations regarding the evaluation of object detection models, the first thing that should be noticed is that the images crafted via Foolbox are in the size of

224x224 and not in the size that they appear in COCO 2017 dataset. Because of this, the obtained AP results of adversarial datasets could not be compared to the results listed in Table 1. To obtain comparable results, a dataset containing coco-original images in the size of 224x224 was created. However, when reducing the size of images to 224x224 the quality of images suffers quite heavily which affects the AP results as seen in Table 4. Because of these changes, I decided to observe the percentual drop that happens in AP metrics between adversarial datasets and the coco-224 dataset when models are evaluated using the evaluation script. This could give approximate results of how the adversarial perturbation would affect the detection accuracy of the model if the images of adversarial datasets would be the same size as the images in the coco-original dataset.

In this research, the evaluation results were based on the primary challenge metric from COCO detection evaluation metrics while other metrics produced by the evaluation script were not discussed when reviewing the results. This decision was made on purpose because otherwise the presentation and analysis would become too broad for the scope of this thesis, even though these results could have formed a better understanding of the models. To get a more informative view of the detections made from an image containing adversarial perturbation, some other metric could have provided better results. When analyzing the amount of perturbation in images, the PSNR value can also be challenging because it differs between images even though a similar amount of perturbation would be included. Also, when calculating the average PSNR value of adversarial datasets, the value with the FGSM attack was lower than with the PGD attack on the same epsilon value. To compare the perturbations created between attacks, a higher epsilon value could have been used with the PGD attack to obtain PSNR values closer to ones with the FGSM attack.

The evaluations were made by analyzing the Mean Average Precision metric that is used in COCO detection evaluation metrics. Apart from the COCO AP metric, the AUC (area under the curve) and ROC (Receiver Operating Characteristic) curve was tried to be calculated to obtain additional information on model performance. AUC-ROC curve is used to calculate the performance of the model in classification problems. When obtaining the raw detection results of datasets made by the object detection models, I noticed that to calculate the AUC-ROC curve, plenty of time would be taken to process the raw results into a cor-

rect shape. Because of the time restrictions, the AUC-ROC curve was dropped from the scope of the thesis. Here are some observations which lead to this decision. The COCO dataset is annotated in a way that not all objects in images are marked in the ground truths. In some images, one large bounding box covers multiple same class objects, meaning that these objects are annotated as one object. The predictions made by COCO API are provided in a format that should be further processed to calculate the AUC-ROC curve. The ground truths in datasets would also need to be converted into the same format to make comparisons possible. The COCO dataset contains objects from 80 classes and for each image the predictions should be converted to a binary problem. If the image contains multiple same class objects close to each other, mapping the correct ground truth and prediction would be difficult.

To discuss the limitations relating to the hardware used in the research, the laptop that I used at first did not have sufficient resources to run evaluations on larger models. The evaluations were then completed on Google Colab in the cloud. However, the free tier of Google Colab had to be used because the Colab Pro is not yet available in Finland. The downside of using the free tier was the GPU limitations, which were broken a couple of times due to running multiple evaluations. This meant that GPU was not then available for a certain period which slowed down the process of running the evaluations.

Future research in this area could relate to evaluating the most recent lightweight models outside of the Tensorflow 2 Detection Model Zoo and models in the YOLO family. This would give a broader view of how lightweight models react to adversarial examples and build a bigger picture of their vulnerability against subtle perturbations. The other adversarial example crafting methods mentioned in chapter 4.5 could be used to test the sensitivity of the lightweight models, while another dataset could be used to train and test the models in the first place. Evaluation of the models could be done by using some other metric than COCO AP or the full list of COCO detection metrics from Figure 11 could be inspected. It would be also beneficial to test how the defense methods mentioned in chapter 4.4 reduce the effects of adversarial perturbation. One last suggestion for future research would be to develop a proper library for testing object detection models against different adversarial examples, as to the best of my knowledge, there currently are none.

9 CONCLUSION

This chapter reflects the overall thesis process. The main goal of the thesis was to study the sensitivity of lightweight object detection models against adversarial perturbation. This was done by selecting a group of pre-trained object detection models from TensorFlow 2 Detection Model Zoo and evaluating their detection accuracies on adversarially crafted datasets. The crafting process of adversarial datasets was completed by using the Foolbox python library and COCO 2017 validation dataset. For adversarial attack type, Foolbox implementations of popular attacks Fast Gradient Sign Methods and The Projected Gradient Descent were selected. While Foolbox did not provide straight implementations for attacking TensorFlow-based object detection models, I chose to craft adversarial examples against the image classification model, combine the output images as a new dataset, and test the object detection models accuracy with this dataset. For evaluation, the COCO Mean Average Precision metric was used. When comparing to the results reported in the literature where adversarial examples are created specifically against the object detection model, like in research made by Chow et al. (2020), the obtained results were not as good with this approach. This is also noticed in research made by Wang, Wang, et al. (2020), who state that the effect of image classification attacks is far from good enough in fooling object detectors. Nevertheless, some promising results were obtained.

The research questions in the thesis were the following:

1. What kind of adversarial attacks currently exists against object detection models?
2. How do the lightweight object detection models react to adversarial examples?

To answer the first research question, a literature review of studies related to adversarial examples in deep neural networks and object detection was conducted. A summary of the findings can be reviewed in chapters 4.3 and 4.5, while in the experiments, the FGSM and the PGD attacks were used to craft adversarial examples. While these attacks were optimized to craft adversarial examples against image classification, they were able to reduce the accuracy of object detectors used in this research when compared to results obtained with clean images.

To answer the second research question, the experiments made in this thesis were evaluated using the proven evaluation metrics. The results show that when applying the adversarial perturbation to the images, the mean average precision used to calculate the accuracy of the object detection model was decreased. This was the case with each model used in the research against each adversarial dataset. Some of the models were able to resist the perturbation more than the others, but in each case, the accuracy of detection suffered. Even though the evaluations could not be made straight with the coco-original dataset, because of the size difference between images, the comparison between adversarial datasets and the coco-224 dataset showed that when the amount of adversarial perturbation is increased in images, the accuracy of detections is decreased.

To get more precise results, adversarial examples should have been crafted using a method specifically optimized to fool an object detection task. The evaluated object detection model should be used as a parameter for the crafting, and the size of produced adversarial examples should be the same as the original COCO dataset to get more universal results. As part of this research, I tried to find a python library that would be capable to craft multiple adversarial attacks against object detection models. However, one specifically for the object detection task was not found and therefore adversarial examples were created against the image classification task. While TensorFlow 2 Detection Model Zoo does not contain the latest object detection models, it still has a quite good collection of models that could be tested. The lightweight models used in this research do not fully reflect the usability of the models in low computation tasks, and it should be stated that the pre-trained models are not designed to be used in critical detection tasks. However, identical models are currently used in some android applications as shown by Y. Huang et al. (2021), which means that the models' vulnerability should be further researched, and their robustness should be improved. These observations create a need for further studies around the research area and the possible development of a python library, for crafting attacks and evaluating the robustness of object detection models.

BIBLIOGRAPHY

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., … Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/

Akhtar, N., & Mian, A. (2018). Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. IEEE Access, 6, 14410–14430. https://doi.org/10.1109/ACCESS.2018.2807385

Allen, M. (2017). The SAGE Encyclopedia of Communication Research Methods. SAGE Publications, Inc. https://doi.org/10.4135/9781483381411

Athalye, A., Engstrom, L., Ilyas, A., & Kwok, K. (2017). Synthesizing Robust Adversarial Examples. ArXiv.Org. http://arxiv.org/abs/1707.07397

Baluja, S., & Fischer, I. (2017). Adversarial Transformation Networks: Learning to Generate Adversarial Examples. ArXiv.Org. http://arxiv.org/abs/1703.09387

Biggio, B., & Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition, 84, 317–331. https://doi.org/10.1016/j.patcog.2018.07.023

Brown, T. B., Mané, D., Roy, A., Abadi, M., & Gilmer, J. (2017). Adversarial Patch. ArXiv.Org. http://arxiv.org/abs/1712.09665

Brownlee, J. (2019). A Gentle Introduction to Object Recognition With Deep Learning. https://machinelearningmastery.com/object-recognition-with-deep-learning/

Carlini, N., & Wagner, D. (2016). Towards Evaluating the Robustness of Neural Networks. ArXiv.Org. http://arxiv.org/abs/1608.04644

Chow, K.-H., Liu, L., Loper, M., Bae, J., Gursoy, M. E., Truex, S., Wei, W., & Wu, Y. (2020). Adversarial Objectness Gradient Attacks in Real-time Object Detection Systems. 2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent

Systems and Applications (TPS-ISA), 263–272. https://doi.org/10.1109/TPS-ISA50397.2020.00042

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 1, 886–893 vol. 1. https://doi.org/10.1109/CVPR.2005.177

Dalvi, N., Domingos, P., Mausam, Sanghai, S., & Verma, D. (2004). Adversarial classification. Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 99–108. https://doi.org/10.1145/1014052.1014066

Dixit, A. A., & Phadke, A. C. (2013). De-noising of Gaussian noise affected images by Non-Local Means algorithm. 2013 International Conference on Circuits, Power and Computing Technologies (ICCPCT), 1215–1218. https://doi.org/10.1109/ICCPCT.2013.6528970

Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q., & Tian, Q. (2019). CenterNet: Keypoint Triplets for Object Detection. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 6568–6577. https://doi.org/10.1109/ICCV.2019.00667

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Tramer, F., Prakash, A., Kohno, T., & Song, D. (2018). Physical Adversarial Examples for Object Detectors. ArXiv.Org. http://arxiv.org/abs/1807.07769

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., & Song, D. (2017). Robust Physical-World Attacks on Deep Learning Models. ArXiv.Org. http://arxiv.org/abs/1707.08945

Girshick, R. (2015). Fast R-CNN. ArXiv.Org. http://arxiv.org/abs/1504.08083

Goodfellow, I., Shlens, J., & Szegedy, C. (2014). Explaining and Harnessing Adversarial Examples. ArXiv.Org. http://arxiv.org/abs/1412.6572

Goodfellow, Ian., Bengio, Yoshua., & Courville, Aaron. (2016). Deep Learning. MIT Press. http://www.deeplearningbook.org

Google. (2021a). Lookout by Google (Vol. 2021, Issue June 15,). https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.reveal&hl=en_US&gl=US

Google. (2021b). Case Study: Adidas uses Object Detection & Tracking for augmented (Vol. 2021, Issue June 15,). https://developers.google.com/ml-kit/case-studies/adidas

Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., & Xu, C. (2019). GhostNet: More Features from Cheap Operations. http://arxiv.org/abs/1911.11907

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778. https://doi.org/10.1109/CVPR.2016.90

Hongkun, Y., Chen, C., Xianzhi, D., Yeqing, L., Abdullah, R., Le, H., Pengchong, J., Fan, Y., Frederick, L., Jaeyoun, K., & Jing, L. (2020). TensorFlow Model Garden.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. v., & Adam, H. (2019). Searching for MobileNetV3. ArXiv.Org. http://arxiv.org/abs/1905.02244

Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. ArXiv.Org. http://arxiv.org/abs/1704.04861

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., & Murphy, K. (2016). Speed/accuracy trade-offs for modern convolutional object detectors. http://arxiv.org/abs/1611.10012

Huang, Y., Hu, H., & Chen, C. (2021). Robustness of on-device Models: Adversarial Attack to Deep Learning Models on Android Apps. http://arxiv.org/abs/2101.04401

Hui, J. (2018). mAP (mean Average Precision) for Object Detection. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. ArXiv.Org. http://arxiv.org/abs/1602.07360

Jiao, L., Zhang, F., Liu, F., Yang, S., Li, L., Feng, Z., & Qu, R. (2019). A Survey of Deep Learning-Based Object Detection. IEEE Access, 7, 128837–128868. https://doi.org/10.1109/ACCESS.2019.2939201

Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial examples in the physical world. ArXiv.Org. http://arxiv.org/abs/1607.02533

Li, F.-F., Karpathy, A., & Johnson, J. (2016). CS231n: Convolutional Neural Networks for Visual Recognition. Lecture 8: Spatial Localization and Detection. http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

Li, X., Lu, R., Liang, X., Shen, X., Chen, J., & Lin, X. (2011). Smart community: an internet of things application. IEEE Communications Magazine, 49(11), 68–75. https://doi.org/10.1109/MCOM.2011.6069711

Li, Y., Tian, D., Chang, M.-C., Bian, X., & Lyu, S. (2018). Robust Adversarial Perturbation on Deep Proposal-based Models. ArXiv.Org. http://arxiv.org/abs/1809.05962

Lienhart, R., & Maydt, J. (2002). An extended set of Haar-like features for rapid object detection. Proceedings. International Conference on Image Processing, 1, I-900-I–903. https://doi.org/10.1109/ICIP.2002.1038171

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2014). Microsoft COCO: Common Objects in Context. http://arxiv.org/abs/1405.0312

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2017). COCO 2017 Validation Dataset. https://cocodataset.org/#download

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M. (2020). Deep Learning for Generic Object Detection: A Survey. International Journal of Computer Vision, 128(2), 261–318. https://doi.org/10.1007/s11263-019-01247-4

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2015). SSD: Single Shot MultiBox Detector. ArXiv.Org. https://doi.org/10.1007/978-3-319-46448-0_2

Liu, X., Yang, H., Liu, Z., Song, L., Li, H., & Chen, Y. (2018). DPatch: An Adversarial Patch Attack on Object Detectors. ArXiv.Org. http://arxiv.org/abs/1806.02299

Liu, Y., Chen, X., Liu, C., & Song, D. (2016). Delving into Transferable Adversarial Examples and Black-box Attacks. ArXiv.Org. http://arxiv.org/abs/1611.02770

Lowd, D., & Meek, C. (2005). Adversarial learning. Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, 641–647. https://doi.org/10.1145/1081870.1081950

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision, 60(2), 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94

Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (2018). ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. ArXiv.Org. http://arxiv.org/abs/1807.11164

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards Deep Learning Models Resistant to Adversarial Attacks. ArXiv.Org. http://arxiv.org/abs/1706.06083

Mathworks. (n.d.). Convolutional Neural Network. Retrieved May 17, 2022, from https://se.mathworks.com/discovery/convolutional-neural-network-matlab.html

Mäyrä, V.-M. (2022). Project AE - Thesis Research Repository. https://github.com/vikamayr/project_ae

Molnar, C. (2019). Interpretable Machine Learning. https://christophm.github.io/interpretable-ml-book/

Moore, B. E., & Corso, J. J. (2020). FiftyOne. https://github.com/Voxel51/Fiftyone

Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., & Frossard, P. (2016). Universal adversarial perturbations. ArXiv.Org. http://arxiv.org/abs/1610.08401

Moosavi-Dezfooli, S.-M., Fawzi, A., & Frossard, P. (2016). DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2574–2582. https://doi.org/10.1109/CVPR.2016.282

Nguyen, A., Yosinski, J., & Clune, J. (2014). Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. ArXiv.Org. http://arxiv.org/abs/1412.1897

Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I. M., & Edwards, B. (2018). Adversarial Robustness Toolbox v1.0.0. CoRR, 1807.01069. http://arxiv.org/abs/1807.01069

Ota, K., Dao, M. S., Mezaris, V., & Natale, F. G. B. de. (2017). Deep Learning for Mobile Multimedia. ACM Transactions on Multimedia Computing, Communications, and Applications, 13(3s), 1–22. https://doi.org/10.1145/3092831

Padilla, R., Passos, W. L., Dias, T. L. B., Netto, S. L., & da Silva, E. A. B. (2021). A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. Electronics, 10(3). https://doi.org/10.3390/electronics10030279

Papernot, N., McDaniel, P., & Goodfellow, I. (2016). Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. http://arxiv.org/abs/1605.07277

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., & Swami, A. (2016). Practical Black-Box Attacks against Machine Learning. https://arxiv.org/abs/1602.02697

Prabhakar, T., Jagan Naveen, V., Prasanthi, A. L., & Vijayasanthi, G. (2011). Image Compression Using DCT and Wavelet Transformations. International Journal of Signal Processing, Image Processing and Pattern Recognition (IJSIP) Korea, Vol.4, pages.61-74. https://www.researchgate.net/publication/221958210_Image_Compression_Using_DCT_and_Wavelet_Transformations

Rauber, J., Zimmermann, R., Bethge, M., & Brendel, W. (2017). Foolbox: A Python toolbox to benchmark the robustness of machine learning models. http://arxiv.org/abs/1707.04131

Rauber, J., Zimmermann, R., Bethge, M., & Brendel, W. (2020). Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX. Journal of Open Source Software, 5(53), 2607. https://doi.org/10.21105/joss.02607

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. ArXiv.Org. http://arxiv.org/abs/1506.02640

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6), 1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

Sara, U., Akter, M., Uddin, M. S., Sara, U., Akter, M., & Uddin, M. S. (2019). Image Quality Assessment through FSIM, SSIM, MSE and PSNR—A Comparative Study. Journal of Computer and Communications, 07(03), 8–18. https://doi.org/10.4236/jcc.2019.73002

Serban, A., Poll, E., & Visser, J. (2020). Adversarial Examples on Object Recognition: A Comprehensive Survey. http://arxiv.org/abs/2008.04094

Soviany, P., & Ionescu, R. T. (2018). Optimizing the Trade-Off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction. 2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 209–214. https://doi.org/10.1109/SYNASC.2018.00041

Srndic, N., & Laskov, P. (2013). Detection of Malicious PDF Files Based on Hierarchical Document Structure. 20th Annual Network and Distributed System Security Symposium, {NDSS} 2013, San Diego, California, USA, February 24-27, 2013. https://www.ndss-symposium.org/ndss2013/detection-malicious-pdf-files-based-hierarchical-document-structure

Su, J., Vargas, D. V., & Sakurai, K. (2019). One Pixel Attack for Fooling Deep Neural Networks. In - IEEE Transactions on Evolutionary Computation (Vol. 23, Issue 5, pp. 828–841). https://doi.org/10.1109/TEVC.2019.2890858

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2014). Intriguing properties of neural networks. https://arxiv.org/abs/1312.6199

Tan, M., & Le, Q. v. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. CoRR, abs/1905.11946. http://arxiv.org/abs/1905.11946

Tensorflow. (2018). Highlights from the TensorFlow Developer Summit, 2018. https://blog.tensorflow.org/2018/03/highlights-from-tensorflow-developer-summit-2018.html

Thys, S., van Ranst, W., & Goedemé, T. (2019). Fooling automated surveillance cameras: adversarial patches to attack person detection. ArXiv.Org. http://arxiv.org/abs/1904.08653

Vasuki, A., & Govindaraju, S. (2017). Deep neural networks for image classification. In Deep Learning for Image Processing Applications (pp. 27–49). IOS Press. https://doi.org/10.3233/978-1-61499-822-8-27

Viola, P., & Jones, M. J. (2004). Robust Real-Time Face Detection. International Journal of Computer Vision, 57(2), 137–154. https://doi.org/10.1023/B:VISI.0000013087.49260.fb

Wang, Y., Tan, Y., Zhang, W., Zhao, Y., & Kuang, X. (2020). An adversarial attack on DNN-based black-box object detectors. Journal of Network and Computer Applications, 161, 102634. https://doi.org/10.1016/j.jnca.2020.102634

Wang, Y., Wang, K., Zhu, Z., & Wang, F.-Y. (2020). Adversarial attacks on Faster R-CNN object detector. Neurocomputing, 382, 87–95. https://doi.org/10.1016/j.neucom.2019.11.051

Wei, X., Liang, S., Chen, N., & Cao, X. (2018). Transferable Adversarial Attacks for Image and Video Object Detection. ArXiv.Org. http://arxiv.org/abs/1811.12641

Wu, F., Xiao, L., Yang, W., & Zhu, J. (2020). Defense against adversarial attacks in traffic sign images identification based on 5G. EURASIP Journal on Wireless Communications and Networking, 2020(1), 173. https://doi.org/10.1186/s13638-020-01775-5

Wu, X., Sahoo, D., & Hoi, S. C. H. (2020). Recent advances in deep learning for object detection. Neurocomputing, 396, 39–64. https://doi.org/10.1016/j.neucom.2020.01.085

Xia, F., Yang, L. T., Wang, L., & Vinel, A. (2012). Internet of Things. International Journal of Communication Systems, 25(9), 1101–1102. https://doi.org/https://doi.org/10.1002/dac.2417

Xie, C., Wang, J., Zhang, Z., Zhou, Y., Xie, L., & Yuille, A. (2017). Adversarial Examples for Semantic Segmentation and Object Detection. ArXiv.Org. http://arxiv.org/abs/1703.08603

Yuan, X., He, P., Zhu, Q., & Li, X. (2019). Adversarial Examples: Attacks and Defenses for Deep Learning. In - IEEE Transactions on Neural Networks and Learning Systems (Vol. 30, Issue 9, pp. 2805–2824). https://doi.org/10.1109/TNNLS.2018.2886017

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 6848–6856. https://doi.org/10.1109/CVPR.2018.00716

Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object Detection With Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems, 30(11), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865

Zhiqiang, W., & Jun, L. (2017). A review of object detection based on convolutional neural network. 2017 36th Chinese Control Conference (CCC), 11104–11109. https://doi.org/10.23919/ChiCC.2017.8029130

Zou, X. (2019). A Review of Object Detection Techniques. - 2019 International Conference on Smart Grid and Electrical Automation (ICSGEA), 251–254. https://doi.org/10.1109/ICSGEA.2019.00065

APPENDICES

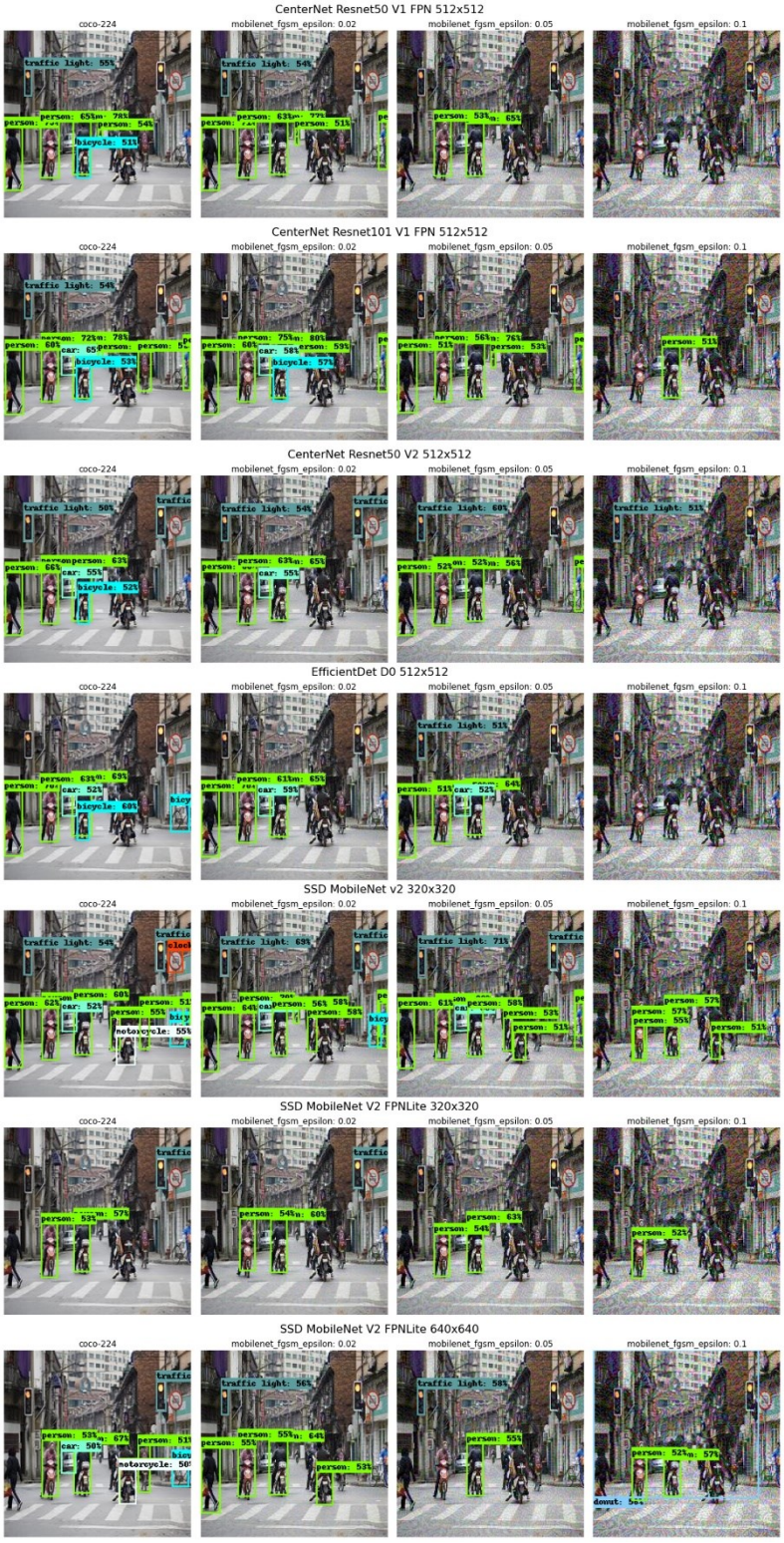
A Model AP metrics and dataset PSNR values

Epsilon value 0.02	CenterNet Resnet101 V1 FPN 512x512	CenterNet Resnet50 V1 FPN 512x512	CenterNet Resnet50 V2 512x512	EfficientDet D0 512x512	SSD MobileNet v2 320x320	SSD MobileNet V2 FPNLite 320x320	SSD MobileNet V2 FPNLite 640x640	
Speed (ms)	34,0	27,0	27,0	39,0	19,0	22,0	39,0	
Tensorflow COCO (mAP)	34,2	31,2	29,5	33,6	20,2	22,2	28,2	
coco-original	34,2	31,2	29,5	33,6	20,2	22,2	28,1	
coco-224	24,1	20,5	19,8	26,9	18,1	18,7	19,2	**PSNR avg.**
MN-0.02-FGSM	20,5	17,1	16,7	24,0	15,9	15,8	15,2	28,20
MN-0.02-PGD	22,2	18,7	18,2	25,7	17,4	17,5	17,3	29,08
RN-0.02-FGSM	21,0	17,4	16,8	24,3	16,3	16,2	15,8	28,14
RN-0.02-PGD	22,3	18,8	18,0	25,9	17,5	17,7	17,4	29,07
EN-0.02-FGSM	20,0	16,7	16,1	23,2	15,2	15,4	15,0	28,18
EN-0.02-PGD	21,2	17,6	17,1	24,7	16,5	16,5	16,0	28,86

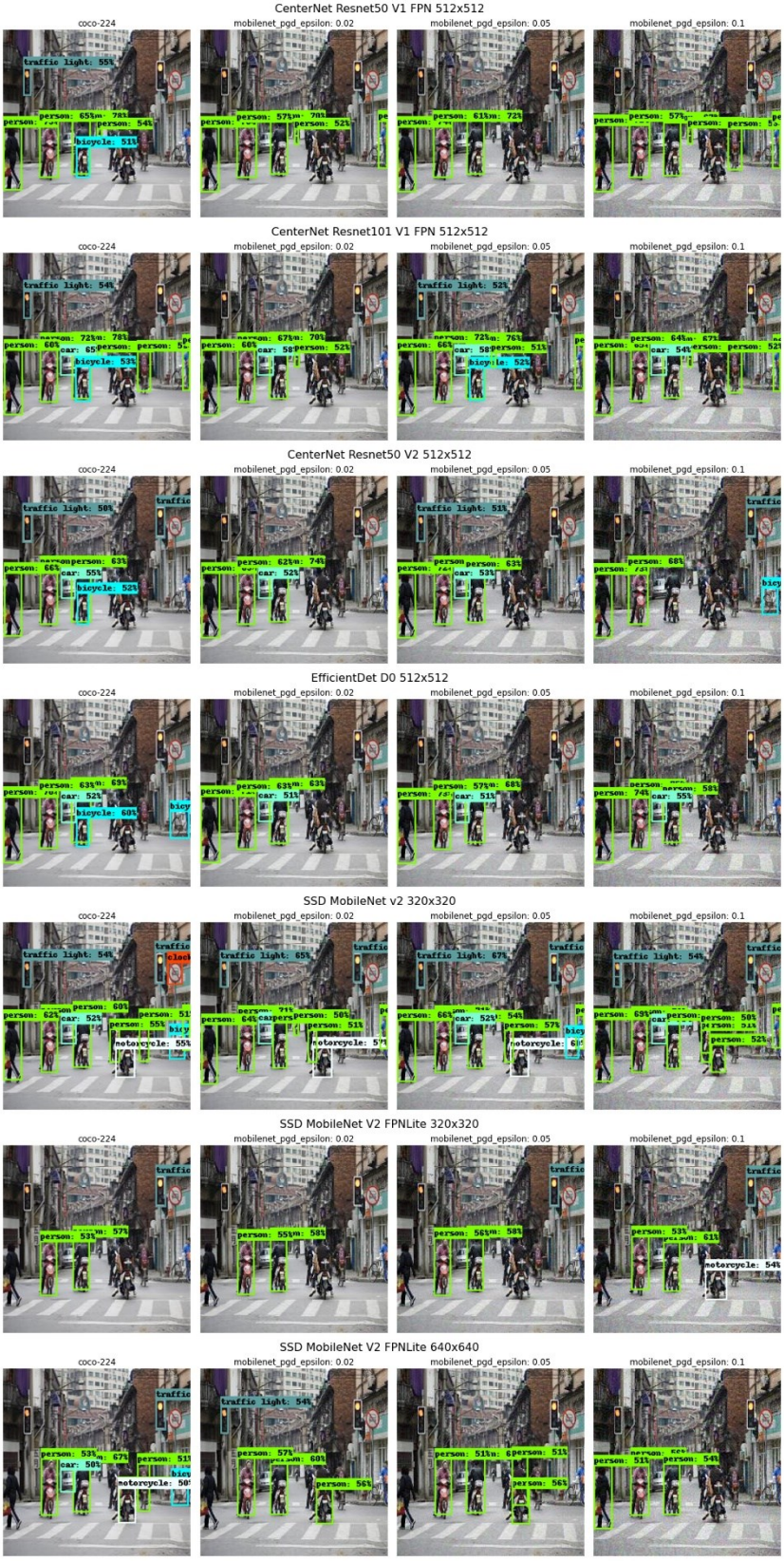
Epsilon value 0.05	CenterNet Resnet101 V1 FPN 512x512	CenterNet Resnet50 V1 FPN 512x512	CenterNet Resnet50 V2 512x512	EfficientDet D0 512x512	SSD MobileNet v2 320x320	SSD MobileNet V2 FPNLite 320x320	SSD MobileNet V2 FPNLite 640x640	
Speed (ms)	34,0	27,0	27,0	39,0	19,0	22,0	39,0	
Tensorflow COCO (mAP)	34,2	31,2	29,5	33,6	20,2	22,2	28,2	
coco-original	34,2	31,2	29,5	33,6	20,2	22,2	28,1	
coco-224	24,1	20,5	19,8	26,9	18,1	18,7	19,2	**PSNR avg.**
MN-0.05-FGSM	13,8	10,6	10,9	17,2	11,1	10,6	9,2	24,88
MN-0.05-PGD	20,5	16,8	16,6	24,0	16,3	15,9	15,3	27,89
RN-0.05-FGSM	13,8	11,0	11,0	17,6	11,6	11,2	9,4	24,85
RN-0.05-PGD	20,6	16,9	16,5	24,2	16,3	16,0	15,4	27,87
EN-0.05-FGSM	13,3	10,4	10,7	16,4	10,7	10,3	9,0	25,03
EN-0.05-PGD	18,9	15,6	15,3	22,6	15,0	14,6	13,8	27,69

Epsilon value 0.1	CenterNet Resnet101 V1 FPN 512x512	CenterNet Resnet50 V1 FPN 512x512	CenterNet Resnet50 V2 512x512	EfficientDet D0 512x512	SSD MobileNet v2 320x320	SSD MobileNet V2 FPNLite 320x320	SSD MobileNet V2 FPNLite 640x640	
Speed (ms)	34,0	27,0	27,0	39,0	19,0	22,0	39,0	
Tensorflow COCO (mAP)	34,2	31,2	29,5	33,6	20,2	22,2	28,2	
coco-original	34,2	31,2	29,5	33,6	20,2	22,2	28,1	
coco-224	24,1	20,5	19,8	26,9	18,1	18,7	19,2	**PSNR avg.**
MN-0.1-FGSM	5,6	4,0	5,1	7,4	4,8	4,5	3,1	20,45
MN-0.1-PGD	16,2	12,8	12,6	19,9	13,4	12,5	11,2	25,14
RN-0.1-FGSM	5,5	4,0	4,8	7,8	5,1	4,8	3,2	20,44
RN-0.1-PGD	16,2	13,0	12,8	20,1	13,6	12,8	11,4	25,12
EN-0.1-FGSM	5,9	4,2	5,0	8,2	5,2	5,3	3,4	20,62
EN-0.1-PGD	14,8	11,5	11,5	17,9	11,9	10,9	9,9	25,03

B Detections on images adapted from COCO dataset (Lin et al., 2017)

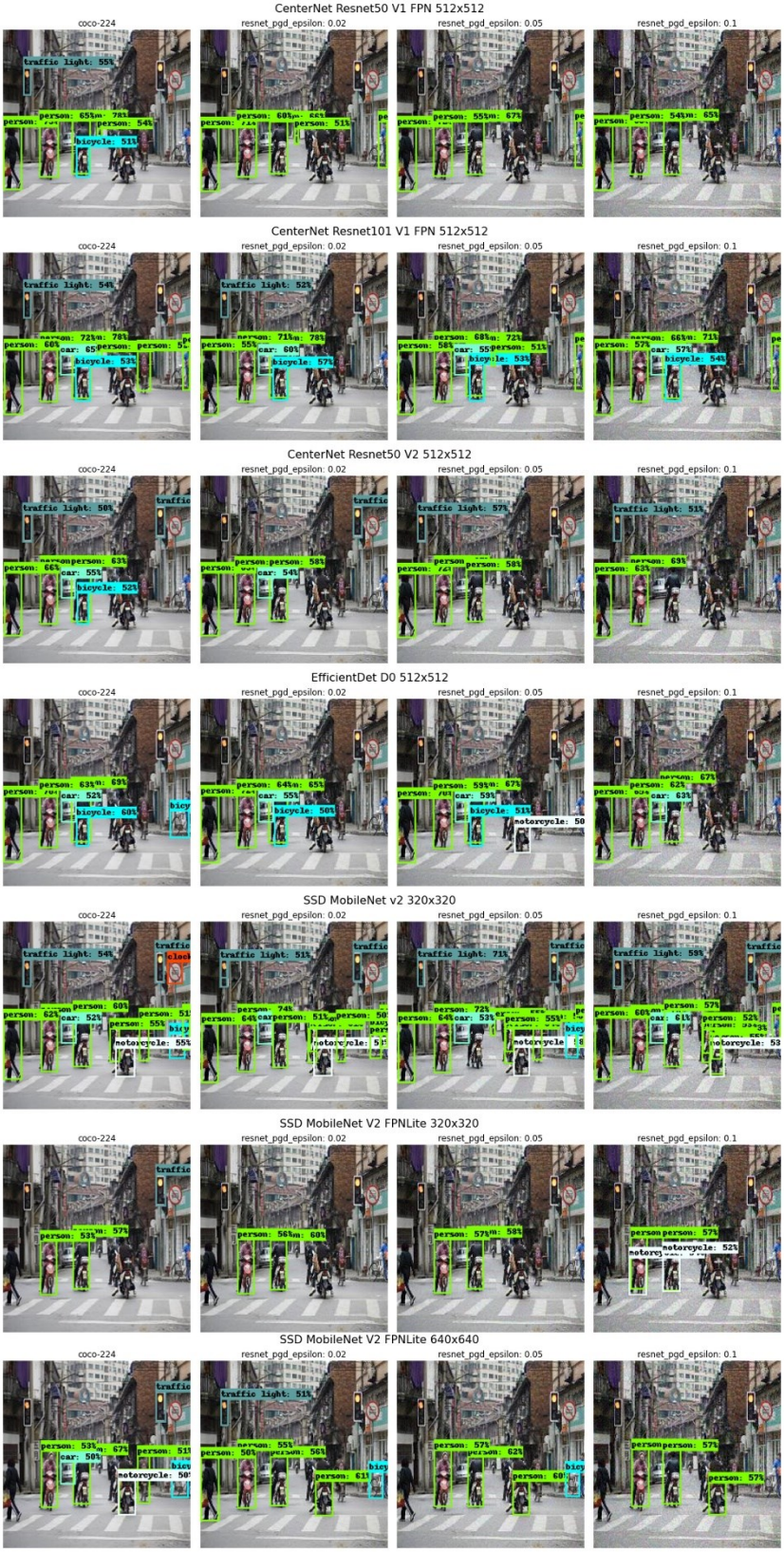


MobileNet FGSM Adversarial Examples



MobileNet PGD Adversarial Examples

ResNet50 FGSM Adversarial Examples



ResNet50 PGD Adversarial Examples

EfficientNet FGSM Adversarial Examples

EfficientNet PGD Adversarial Examples

C License and copyright of Figures and Appendix B

This appendix list figures containing images or annotations adapted or based on some other image or annotation and provides information about their license and copyright. Copyrights of images in the COCO dataset do not belong to COCO Consortium, which is why this appendix was created.

Figure 6

First-row corner images adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000000285.jpg. Original: http://farm8.staticflickr.com/7434/9138147604_c6225224b8_z.jpg. CC BY 2.0: https://creativecommons.org/licenses/by/2.0/

Third-row corner images adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000079408.jpg. Original: http://farm6.staticflickr.com/5324/9734800916_1a66b88f77_z.jpg. CC BY 2.0: https://creativecommons.org/licenses/by/2.0/

Figure 10

Image based on: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000128372.jpg. Original: http://farm9.staticflickr.com/8479/8172792873_5d083ef147_z.jpg. CC BY-SA 2.0: https://creativecommons.org/licenses/by-sa/2.0/

Figure 11

Image based on: COCO detection evaluation metrics (Lin et al., 2014). [Image]. Accessed 18.5.2022 on https://cocodataset.org/#detection-eval. CC BY 4.0: https://creativecommons.org/licenses/by/4.0/

Figure 14

Based on: COCO detection evaluation metrics (Lin et al., 2014) obtained with model_main_tf2.py script of Tensorflow 2 Object Detection API (J. Huang et al., 2016). [Figure]. Accessed 18.5.2022 on evaluation notebook created on this thesis, available on https://github.com/vikamayr/project_ae (Mäyrä, 2022).

Figure 21, Figure 22, Figure 23, Figure 30

Images adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000000139.jpg. Original: http://farm9.staticflickr.com/8035/8024364858_9c41dc1666_z.jpg. CC BY-NC 2.0: https://creativecommons.org/licenses/by-nc/2.0/

Figure 24

First image on first and third row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000000139.jpg. CC BY-NC 2.0: https://creativecommons.org/licenses/by-nc/2.0/

Second image on first and third row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000006471.jpg. Original: http://farm4.staticflickr.com/3228/2755941377_ea852330ca_z.jpg. CC BY-NC SA 2.0: https://creativecommons.org/licenses/by-nc-sa/2.0/

Third image on first and third row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000128372.jpg. Original: http://farm9.staticflickr.com/8479/8172792873_5d083ef147_z.jpg. CC BY-SA 2.0: https://creativecommons.org/licenses/by-sa/2.0/

Fourth image on first and third row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000142472.jpg. Original: http://farm8.staticflickr.com/7274/7871818696_6bf4905675_z.jpg. CC BY 2.0: https://creativecommons.org/licenses/by/2.0/

Fifth image on first and third row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000226417.jpg. Original: http://farm1.staticflickr.com/193/495093205_cbb83a14ff_z.jpg. CC BY-SA 2.0: https://creativecommons.org/licenses/by-sa/2.0/

Figure 25

Left image on first row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000000139.jpg. CC BY-NC 2.0: https://creativecommons.org/licenses/by-nc/2.0/

Middle image on first row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000000285.jpg. Original: http://farm8.staticflickr.com/7434/9138147604_c6225224b8_z.jpg. CC BY 2.0: https://creativecommons.org/licenses/by/2.0/

Right image on first row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000000632.jpg. Original: http://farm2.staticflickr.com/1241/1243324748_eea455da9f_z.jpg. CC BY 2.0: https://creativecommons.org/licenses/by/2.0/

Left image on second row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000000724.jpg. Original: http://farm4.staticflickr.com/3576/3383381911_9d6d5b63a1_z.jpg. CC BY-NC-SA 2.0: https://creativecommons.org/licenses/by-nc-sa/2.0/

Middle image on second row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000000776.jpg. CC BY-NC-SA 2.0: https://creativecommons.org/licenses/by-nc-sa/2.0/

Right image on second row adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000000785.jpg. Original: http://farm8.staticflickr.com/7015/6795644157_f019453ae7_z.jpg. CC BY 2.0: https://creativecommons.org/licenses/by/2.0/

Annotations adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Annotations]. Accessed 18.5.2022 on https://cocodataset.org/#download. CC BY 4.0: https://creativecommons.org/licenses/by/4.0/

Figure 26, Figure 29, Appendix B

Images adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000226417.jpg. Original: http://farm1.staticflickr.com/193/495093205_cbb83a14ff_z.jpg. CC BY-SA 2.0: https://creativecommons.org/licenses/by-sa/2.0/

Figure 28

First row images adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000142472.jpg. Original: http://farm8.staticflickr.com/7274/7871818696_6bf4905675_z.jpg. CC BY 2.0: https://creativecommons.org/licenses/by/2.0/

Second row images adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000226417.jpg. Original: http://farm1.staticflickr.com/193/495093205_cbb83a14ff_z.jpg. CC BY-SA 2.0: https://creativecommons.org/licenses/by-sa/2.0/

Third row images adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000006471.jpg. Original: http://farm4.staticflickr.com/3228/2755941377_ea852330ca_z.jpg. CC BY-NC-SA 2.0: https://creativecommons.org/licenses/by-nc-sa/2.0/

Fourth row images adapted from: COCO 2017 Validation Dataset (Lin et al., 2017). [Image]. Accessed 18.5.2022 on http://images.cocodataset.org/val2017/000000128372.jpg. Original: http://farm9.staticflickr.com/8479/8172792873_5d083ef147_z.jpg. CC BY-SA 2.0: https://creativecommons.org/licenses/by-sa/2.0/