Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Object Detection in Fine Art Photography

Bachelor thesis, spring semester, 2020

Fabian Meyer

Supervisor: Dr. Simone Lionetti

Lucerne University of Applied Sciences and Arts
Bachelor in Information Technology
May 31, 2020

Bachelorarbeit an der Hochschule Luzern - Informatik

Titel: Object Detection in Fine Art Photography

Student: Fabian Meyer

Studiengang: Bachelor Informatik

Abschlussjahr: 2020

Betreuungsperson: Dr. Simone Lionetti

Expertin / Experte: Roman Bachmann, Swisscom

Eidesstattliche Erklärung
Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt habe, alle verwendeten Quellen, Literatur und andere Hilfsmittel angegeben haben, wörtlich oder inhaltlich entnommene Stellen als solche kenntlich gemacht haben, das Vertraulichkeitsinteresse des Auftraggebers wahre und die Urheberrechtsbestimmungen der Fachhochschule Zentralschweiz (siehe Markblatt ńStudentische Arbeitenż auf MyCampus) respektieren werde.

Ort / Datum, Unterschrift: _____

Abgabe der Arbeit auf der Portfolio Datenbank

Bestätigungsvisum Studentin / Student
Ich bestätige, dass ich die Bachelorarbeit korrekt gemäss Merkblatt auf der Portfolio Datenbank abgelegt habe. Die Verantwortlichkeit sowie die Berechtigungen habe ich abgegeben, so dass ich keine Änderungen mehr vornehmen kann oder weitere Dateien hochladen kann.

Ort / Datum, Unterschrift: _____

Verdankung
Danke liebe Freunde

Eingangsvisum (durch das Sekretariat auszufüllen):

Rotkreuz, den _____ Visum: _____

Hinweis: Die Bachelorarbeit wurde von keinem Dozierenden nachbearbeitet. Veröffentlichungen (auch auszugsweise) sind ohne das Einverständnis der Studiengangleitung der Hochschule Luzern Informatik nicht erlaubt.

Abstract

Contents

1 Introduction

Object detection is an important computer vision and machine learning task that consists in the localisation and classification of items within digital images. The business and industry applications of this technology are growing in number and relevance, especially because of the tremendous progress, largely driven by deep learning, that has been made in recent years.

Uses of object detection technologies include applications in security measures for example images from surveillance cameras, search engines, object tracking or counting in traffic, autonomous driving cars, robotics in general and the field of medical- and bioinformatics.

1.1 Task description

The aim of this work is to develop a web-application that is built around an object-detection model. This web-application has to take in a picture, looks for objects in it and has to return a "tidied up" image with the objects found in it. The "tidied up" image is inspired by "Kunst aufräumen", a series of fine-art photographies by Swiss artist Urs Wehrli, that consists of two images, a "messy" one and a "tidy" one. In the "tidy" image, similar objects are shown next to each other, a task that is analogue to classification in computer vision applications. One example picture from "Kunst aufräumen" is shown here:

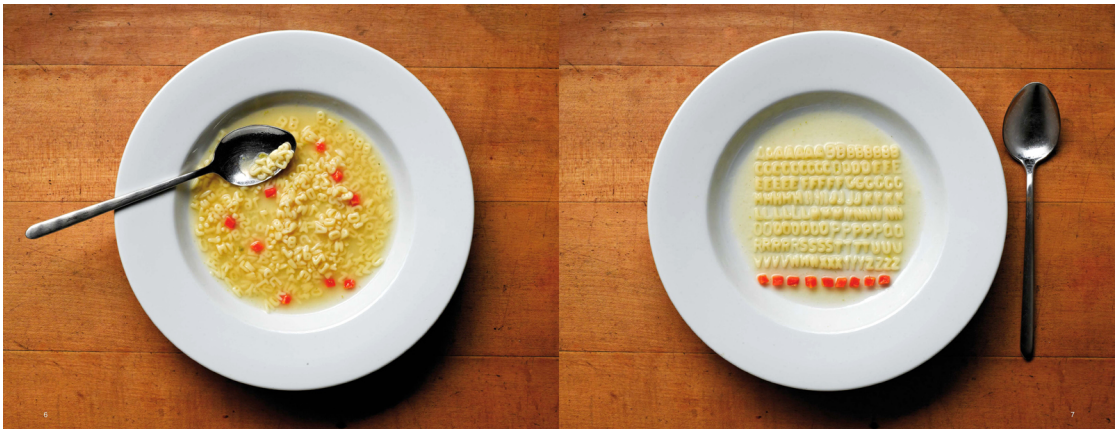

Figure 1.1: Sample image from "Kunst aufräumen" by Urs Wehrli

In a further step, an existing object detection model has to be refined by retraining it on a dataset with images from artistic photography and the performance of the two models have to be examined. One important element in the research of object detection is computation

of scores to evaluate the quality of the model. An additional tasks was to develop an own metric to evaluate the different models when applying them to data of artistic photography.

1.2 Why fine art photography?

Models for object detection are usually trained on datasets, containing pictures that show objects in a very clear manner. These images usually contain only a few objects and are shot in natural lighting with natural colours. Composition-wise they are simple as it is the goal to depict the object in a most clear way.

Figure 1.2: Sample image from COCO dataset

Fine art photography pictures on the other hand is in strong contrast to these images, as they are often depicting objects who do not belong together usually. These images are often shot in studios with artificial lighting and heavy post-processing or even digital manipulation. And they often contain a lot more objects in it which even can overlap each other.

Figure 1.3: Sample image from artist David LaChapelle

The question that now arises is, how do object detection models that are trained on traditional datasets perform on given images of artistic photography?

2 Current state

2.1 A brief introduction into discrete two-dimensional convolution

The model that is used for this project is from the family of convolutional neural networks or CNNs. These are models that are especially useful to perform operations on digital images. CNNs contain at least one convolutional layer that computes a mathematical operation that is called discrete convolution. Discrete convolution operation has been proved useful when applied on digital images to detect structural features like edges and corners. When applying discrete convolution to an image (a two dimensional array of values), a filter with a given two dimensional kernel is used to perform convolution on the image. With the given kernel, a segment that has the same size as the kernel is taken from the input image and an elementwise multiplication between these two matrices is performed. The sum of the results gets written in the output (also called feature map). As most kernels are much smaller than the input image, this inevitable results in smaller feature maps than input images, especially after applying convolution to the same image multiple times. To solve this problem, one can add a padding (increase the image size by adding specific values). An example of a two dimensional convolution operation with padding is shown here:

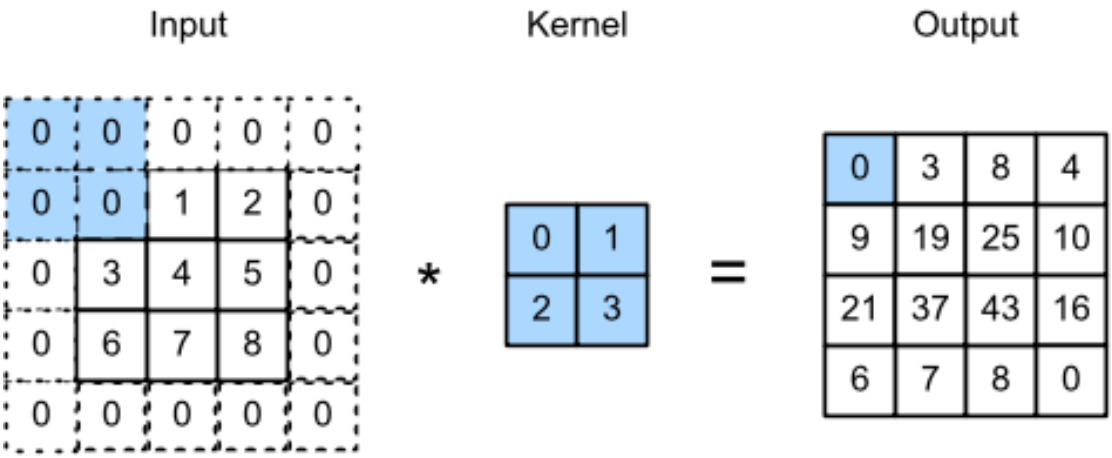

Figure 2.1: Example of a two dimensional convolution operation with padding

After the convolution is computed for a specific segment, the kernel shifts further some steps in the image array, called stride. There exists a lot of different kernels for different purposes. To detect edges and corners there exist specific kernels especially for that kind

of task. For example if one wants to detect vertical edges, a kernel like the following can be used: $P_y = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$, to detect diagonal edges, one can use a kernel like the following: $S_d = \begin{pmatrix} -0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix}$

It is also possible to learn the optimal kernel in the process of neural network training.

2.2 About convolutional neural networks

The advantage of CNNs compared to traditional multi layer perceptrons (MLPs) is that discrete convolutional operation is translational invariant. This means it is irrelevant where in a given image a specific object is located: The model will not learn the position of the specific object. Another huge advantage of CNNs over MLPs is that CNNs usually have sparse interactions. This means that it is possible to yield a good performance even if some adjacent layers are not fully connected with each other. Another way to speed up the training process is to use parameter sharing: Some weigths will be used at multiple places at the same time. These two techniques, sparse interactions and parameter sharing does make it possible to train very deep convolutional neural networks in a feasible amount of time.

It is important to note that when applying discrete convolution on digital images, only primitive features can be extracted. But by combining convolutional layers with other layer types, it is possible to extract much more complex features that can be used to classify objects as humans or animals or vehicles and so on.

Another important kind of layer that is usually employed in CNNs are pooling layers. Also called subsampling or downsampling layers sometimes, they reduce the size of a given image, by computing a statistic metric to summarize multiple values. Some frequently used pooling layers are max- and average-pooling. The advantages of pooling layers are smaller input, parameter reduction and higher invariance to scaling and transformations.

The third important kind of layer is ReLU (rectified linear unit), which computes an activation function used by the neural network. The mere use of this layer is to preprocess the image before the next convolutional operation. This just adjusts the image brightness in a ways that all negative values (values that are darker than the middle grey of an image) got adjusted to middle grey.

The last layer of a CNN contains the number of classes that the network should predict. This layer is usually fully connected to the layer before.

2.3 From CNNs to Mask R-CNN

The very first CNN appeared in the year 1994, it was named LeNet5, after Yann LeCun. This fundamental work was the first network that used convolutional and pooling layers to process images. In a time long before consumer graphic processing units (GPUs), where even CPUs were slow, it was crucial to reduce the number of parameters to a bare minimum (accomplished with sparse connect layers). An image, showing LeNet5 with its different layers and operations in between is shown here:

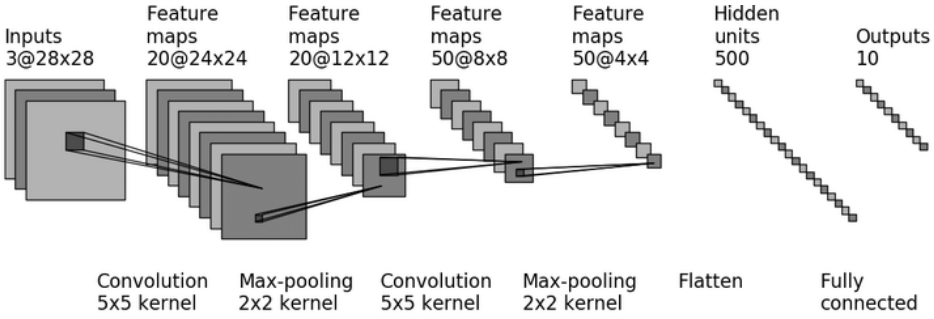

Figure 2.2: Overview of LeNet5 from the year 1994

Starting with AlexNet by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton in 2012, CNNs gained significantly performance gains when applied on image classification tasks. AlexNet took the idea of a CNN from LeNet5 and added more layers to the network and was the first to include ReLU layers. It also used dropout techniques to avoid overfitting during the training process. AlexNet was still an image detection or image classification model, trained to label images. This means a single label (class) is given to a whole input image.

ResNet (residual neural network) in 2015 was the first CNN that included residual blocks or identity shortcut connections. These are connections in the network that skip one or more layers and just use the input value as an output (called mathematical identity). This was another step to reduce computation cost and lead to even deeper networks. ResNet is still used today as part of the backbone in a lot of object detection and instance segmentation models.

With R-CNN (regions with CNN features) in 2013, the rise of object detection models began. These researcher asked themselves, how can one use the techniques from CNNs to not only classify an image but to classify multiple objects in that image? These family of models are capable of labelling multiple objects depicted in the same image with each a bounding box and a class prediction. R-CNN models do that by proposing regions in that potential objects may lie. R-CNN is thus called a two stage model: The first stage scans the image and generates region proposals with the help of a region proposal network (RPN), whereas the second stage uses these regions to extract CNN features from it and to ultimately classify objects in it. For the classification step in the last layer, R-CNN uses a support vector machine (SVM) as a classifier. In the very last step, a regression is used to further tighten the coordinates of the bounding boxes of each object. [1]

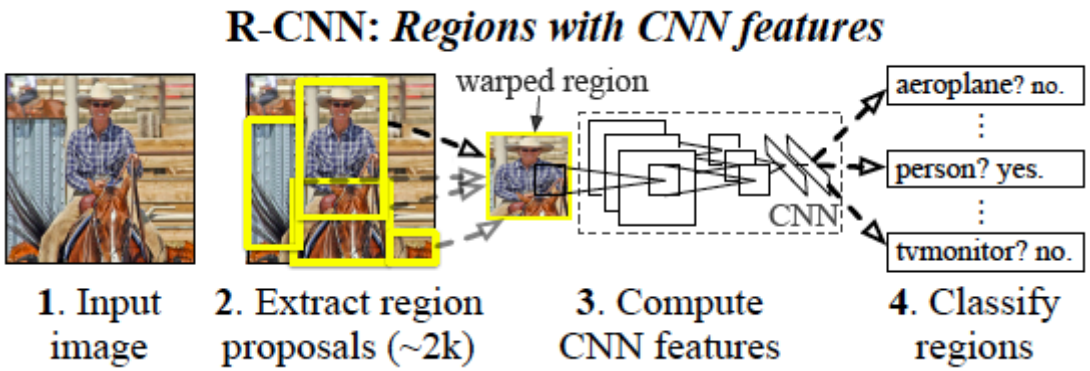

Figure 2.3: Architecture of R-CNN

In 2017, the team from Ross Girshick, one of the creators of R-CNN, delivered Fast R-CNN an improved, faster version of it. In R-CNN there are a lot of overlapping region proposals and every computation gets calculated again, even if the regions are very similar to each other. To circumvent this they invented region of interest pooling (RoIPool). RoIPool shares these computation across the regions of an image and can speed up computation time a lot. The other improvement was to put all computations in a single network (compared to R-CNN where for example classification ran in a single network).

Also in 2017 by the team from Ross Girshik, the second iteration of R-CNN got released, called Faster R-CNN. The main improvement was to use just one CNN that produces a single feature map for both region proposal and classification. Features used by both the first and the second stage of the model can be shared to speed up the inference process.

2.4 Mask R-CNN

With Mask R-CNN from 2017 (also by Ross Girshik et. al) it is possible to not only predict the bounding box of an object but also to predict a mask that shows the exact shape of the object (called pixel level segmentation). These models are called instance segmentation models, because they produce a mask for every object in the image. This is accomplished by adding a branch to the Faster R-CNN model that computes a binary mask for a given object in the image. An overview of Mask R-CNN can be seen here:

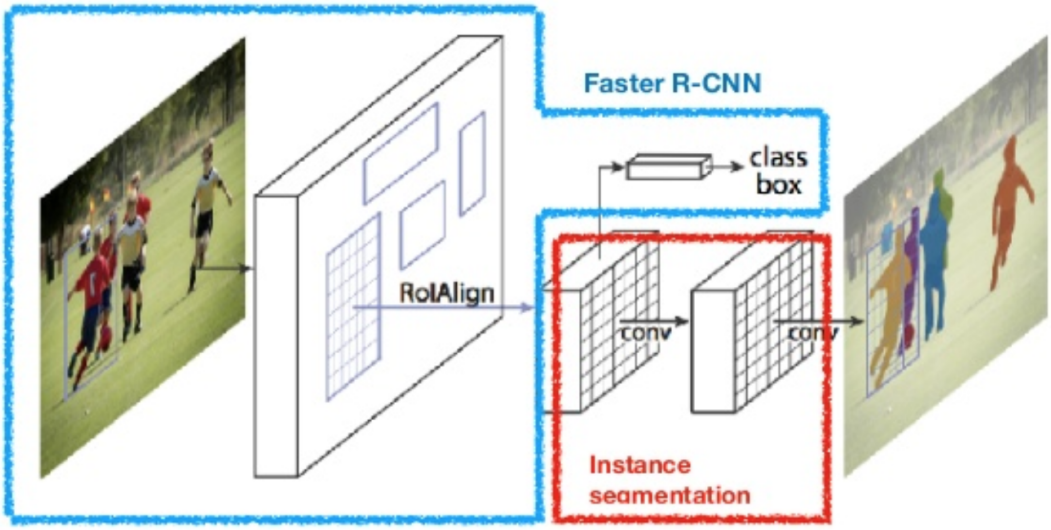

Figure 2.4: An overview of Mask R-CNN

- About feature pyramid networks - About ResNet101 (backbone)

An overview over the different tasks and approaches and their models in the field of object recognition can be seen here:

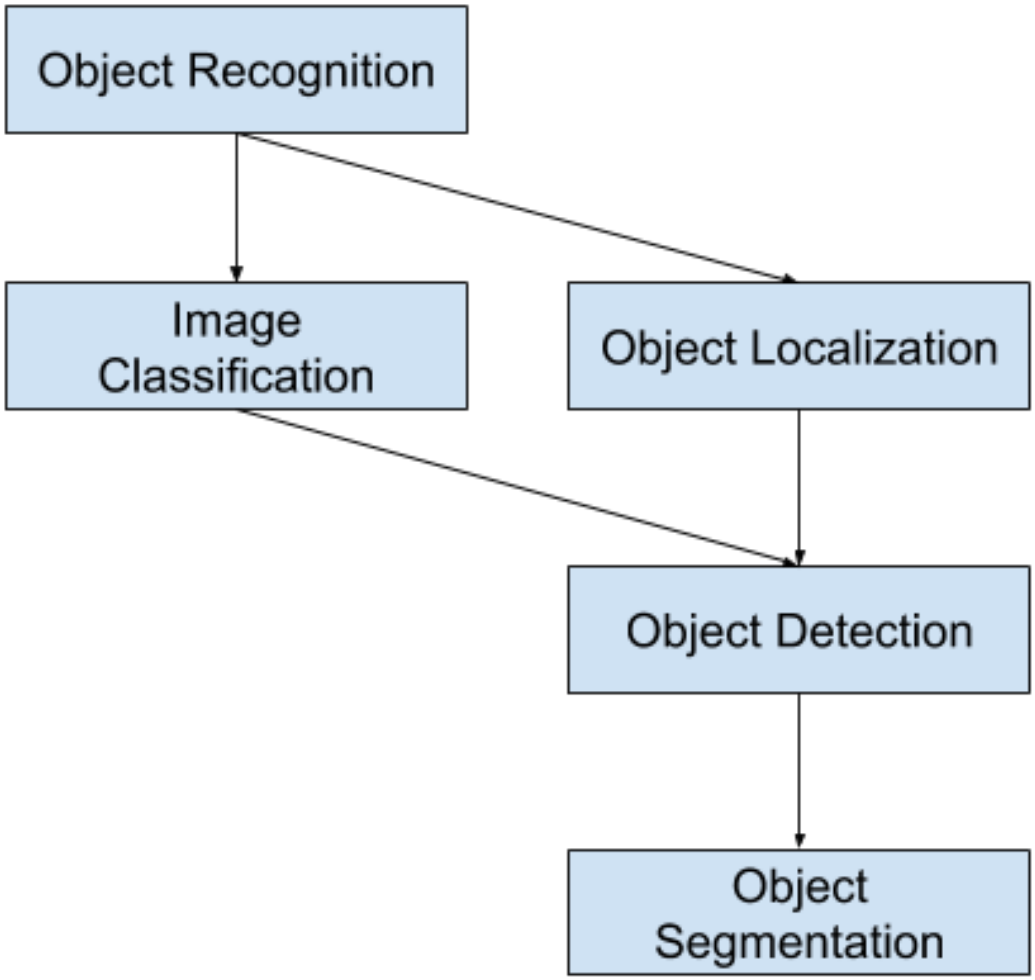

Figure 2.5: Tasks and approaches in object recognition

3 Methods

3.1 Toolchain

The whole project can be summarized in four single steps or tasks:

1. Data: Collect a dataset, for inference, for training and for testing
2. Model: Get to know at least one model, for inference and for retraining
3. Tidy: For a given input picture, return an image that displays a repertoire of all found objects in the input picture
4. App: Turn the former three tasks into an application that can be reached from a web browser.

In order to develop a minimal viable product, all these four tasks have to be solved first. The way of proceeding was to develop a minimal viable product first and then to start with refining the model and adjusting the app. This means going through the full circle first, before adjustments and proceeding into retraining are made.

3.2 Project management

In the beginning of the project, a project management plan had to be developed. The project management plan contains a timeline with all (then known) tasks and a list of all milestones. Trello together with a timeline add-on (a gant chart) was used as project management toool. The project management plan and all tasks and milestones can be seen via this link: `https://trello.com/b/srqnMstX/object-detection-in-fine-art-photography`. A picture, showing the full timeline is shown here:

To better structure the project, a number of milestones have been chosen:

1. Test pretrained models on Google Colab with new data
2. Build web application prototype
3. Finetune model with fine art photography data
4. Test finetuned model on new data
5. Finish web application with new model

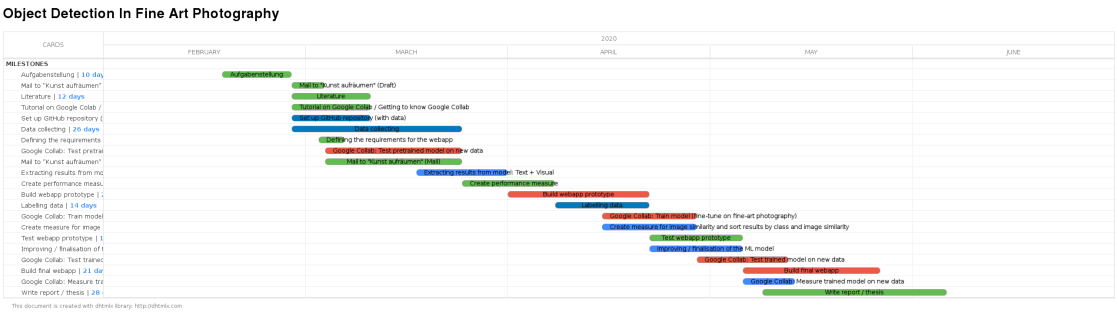

Figure 3.1: Timeline view of the project management plan

3.3 Codebase

To develop the project, three different code repositories have been created: A git repository containing different datasets. This repository was used for testing out different models and frameworks when applying inference on the images. When developing the tidied up images, it was used too. It can be reached here: `https://gitlab.enterpriselab.ch/iameyer/odifap`. Another git repository was created to develop the web application. It can be accessed here: `https://gitlab.enterpriselab.ch/iameyer/odifap-program`. The third git repository was created to serve all necessary files for retraining the models. This repository contains a config.py file for every model and a .json file, containing the bounding boxes and masks for retraining. Large files, like .pth files (checkpoint files, that are generated during training process) were saved to Google drive with the help of a custom Python script.

3.4 Testing

In favour of having more time available for the development cycle and retraining of the model, no testing strategy has been selected.

4 Ideas and concepts

4.1 Basic idea

The basic idea of creating an application that takes in images and returns a repository of objects found in it is original to our best knowledge and leaves much room for interpretation and customization.

4.2 Alternative frameworks and models

As can be seen in the overview graphic of different object recognition tasks in figure 2.5 on page 9, there are multiple possibilities to detect objects in an image. As it gives a nicer output, an instance segmentation model was chosen over an object detection model. Another idea that came to our mind was to use of an object of the family of image captioning models. Image captioning models are capable of generating a text that describes the scene depicted in an input image. These use a CNN in a first step to analyse the image, combined with a generative network to generate a text from it afterwards. Image captioning would have been especially interesting when applied to fine art photography, because many of these pictures show messages that are loaded with irony or socio-critical or political messages.

4.3 Alternative technologies to develop the web application

There exist some different tools to create a web application with Python. The classic way to do this would have been to use Python as a backend programming language, that computes the logic of the application and to use a frontend programming language like JavaScript to develop the user interface. There is also a tool available that can take in a Jupyter notebook and turn it into a running web application, called Voilà. Voilà works with any Jupyter kernel and can also be used to develop an application from other languages than just Python.

4.4 Alternative research questions

The most obvious research quesiton would have been: What happens if a model gets trained on images by artist X and gets tested on images by artist Y, that contain the equal classes of objects in it? It would have been interesting to find out, whether the model will learn the distinct style of the artist, like colours, contrast, camera angle, lighting and so on. Of course another model could have been trained on the corresponding images from artist Y and the two models could have been compared with each other and also with the base (pretrained)

model. As this needs at least two images from different artists that contain the same objects in it, it was rejected. However with a suitable performance metric it would allow to make a concluding statement about object detection in fine art photography.

Another, more humorous task would have been to create something new with the found objects in the image. For example a fruit bowl generator algorithm, that takes in an image, searches for objects of different class of fruits and places them into a bowl and returns the new "fine art"-image. This could be accomplished with just hard-coding the features needed, or even better, with a generative model. As this is a complete different topic this project idea was rejected.

5 Realisation

5.1 Data collection

The first step in this project was to collect some data. Two pictures of fine art photography were already shown in the project description: An image by german photographer Andreas Gursky and one by Swiss artist Urs Wehrli. The main motivation behind data collection was to obtain images from different artists that each show a lot of different objects. This let us examine the performance of an object detection model applied to fine art photography, by using images that contain a lot of objects that optimally are also included in the COCO dataset.

The COCO (Common Objects in COntext) dataset is among the most popular image datasets created for object recognition tasks. It was lanced by Microsoft in 2014 and the most recent version was published in 2017. It uses 80 different classes for object detection tasks that can be seen here:

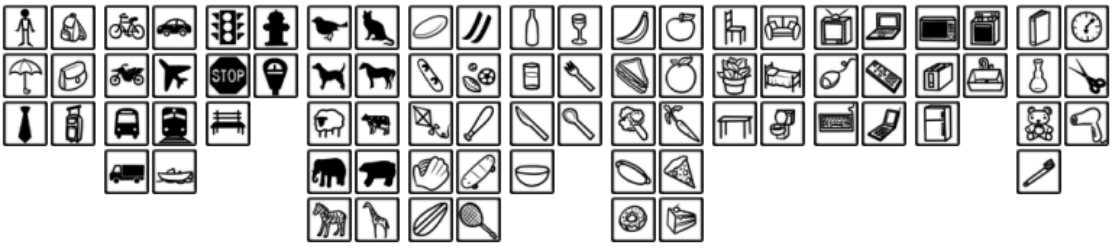

Figure 5.1: All 80 classes from COCO dataset 2017

The COCO dataset not only contains bounding boxes and binary masks for object detection and segmentation, it also includes informations for keypoint estimation, panoptic segmentation and image captioning tasks. It is one of the big baselines for object detection tasks for many years now.

In the end, 42 different images from the four artists Urs Wehrli, Andreas Gursky, David LaChapelle and Jeff Wall has been gathered. Urs Wehrli, the swiss artist behind "Kunst aufräumen" was asked to give his permission to use his images for this project. Thankfully he gave permission to do so. The dataset has then been examined with different object segmentation models on Google Colab.

Google Colab is a free to use infrastructure, powered by Google, that offers a zero-configuration Python and Jupyter notebook environment. Running on Linux Ubuntu with the latest Nvidia GPUs, this is a good option to get started with deep learning, as there are a lot of notebooks about every single kind of neural network tasks available.

5.2 Model selection

With the gathered dataset, different object segmentation models from different frameworks have been tried out on the dataset in inference mode. All these models have been pretrained on the COCO-dataset. The tried out models were: Mask R-CNN, CenterNet, Detectron2 and ShapeMask. In general, Mask R-CNN outperformed the other models, when inference is run on the dataset. Detectron2 also delivered a good performance but it threw an error when running on many images in a loop on Google Colab.

5.3 Framework selection

At the same time different frameworks have been tested out. These were Tensorflow from Google, Detectron from Facebook and MMDetection, which is a part of the OpenMMLab project developed by Multimedia Laboratory by the Chinese University of Hong Kong. All these frameworks do at least contain one Mask R-CNN model. MMDetection has been chosen as the framework to develop the project in, because it worked out-of-the box when tried out on Google Colab. It returned results when running in inference mode on our dataset in about seven Minutes. MMDetection has a vast number of state-of-the-art models for computer vision tasks available and offers a high-level API that speeds up its usage. It is built on top of PyTorch (primarily developed by Facebook) and delivers a very good performance. An overview of available models in MMDetection is shown here:

	MMDetection	maskrcnn-benchmark	Detectron	SimpleDet
Fast R-CNN	✓	✓	✓	✓
Faster R-CNN	✓	✓	✓	✓
Mask R-CNN	✓	✓	✓	✓
RetinaNet	✓	✓	✓	✓
DCN	✓	✓	✓	✓
DCNv2	✓	✓		
Mixed Precision Training	✓	✓		✓
Cascade R-CNN	✓		*	✓
Weight Standardization	✓	*		
Mask Scoring R-CNN	✓	*		
FCOS	✓	*		
SSD	✓			
R-FCN	✓			
M2Det	✓			
GHM	✓			
ScratchDet	✓			
Double-Head R-CNN	✓			
Grid R-CNN	✓			
FSAF	✓			
Hybrid Task Cascade	✓			
Guided Anchoring	✓			
Libra R-CNN	✓			
Generalized Attention	✓			
GCNet	✓			
HRNet	✓			
TridentNet [17]				✓

Figure 5.2: Overview of MMDetection models

5.4 Choosing the programming language

Because most of the deep-learning frameworks are using Python and also of its ease of use, Python has been used as the sole programming language to develop this project. In addition Python does offer a lot of visual computing libraries that were used to create the tidied up image.

5.5 Creating the tidied up image

After the dataset has been collected, and the model and the framework have been chosen, the next step was to extract all single objects from the output of the model when running in inference mode on an image. Mask R-CNN model outputs for every input image among other things a list with with each a list of bounding boxes, masks, confidence score and

predicted class of all objects found in the image. These four results were saved in a list object. The output is per default limited to 100 objects per image. The confidence score was used to filter all objects to get the objects with the highest quality as predicted by the model. The binary mask of the found objects was then applied to the image, creating binary images of the object. An example can be seen here:

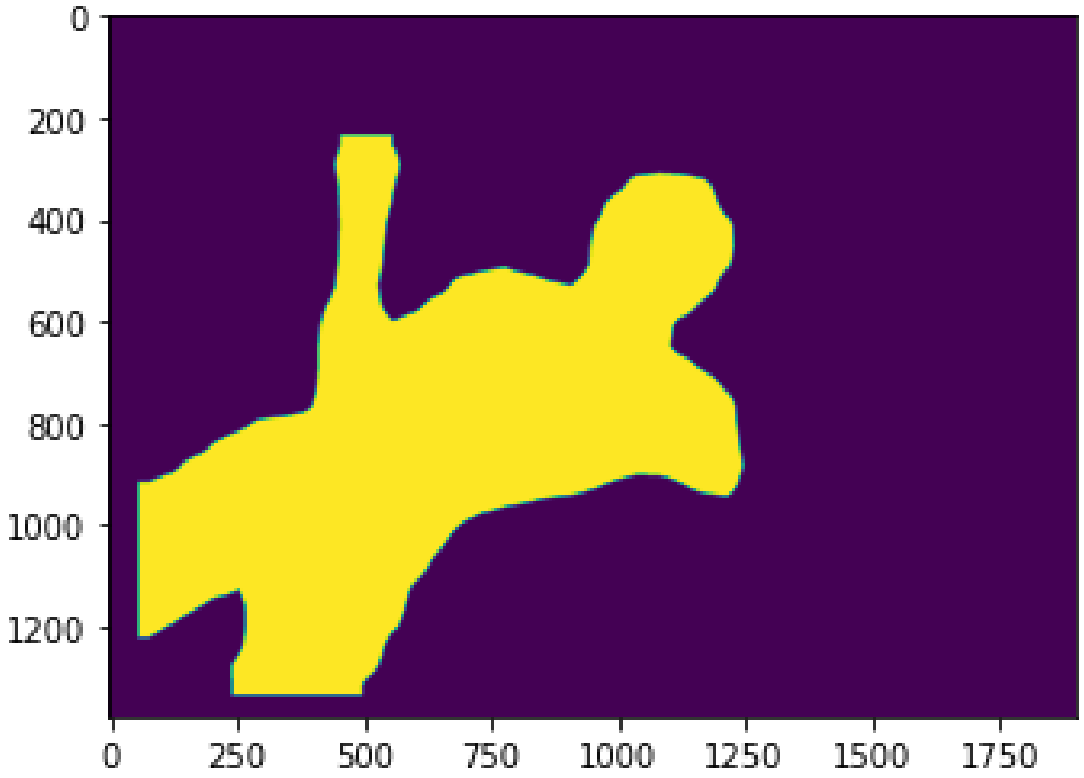

Figure 5.3: Sample binary mask image

With help of the binary masks, the background of the found object was coloured white whereas the foreground stayed as is. Subsequently the corresponding bounding boxes were taken to cut out the objects from the image. The last step is to create a grid with all the found objects. This was done with the help of Python's Matplotlib. The idea is to create a grid with the number of rows according to the number of classes and insert every object as a new column. The problem with using Matplotlib's. `plt.subplots` is that it creates a grid where every image has to have the same size. To circumvent this, a more complicated approach was taken, by creating a grid manually with the help of the width and the height of all the found objects. A sample input of a tidied up image together with its output can be seen here:

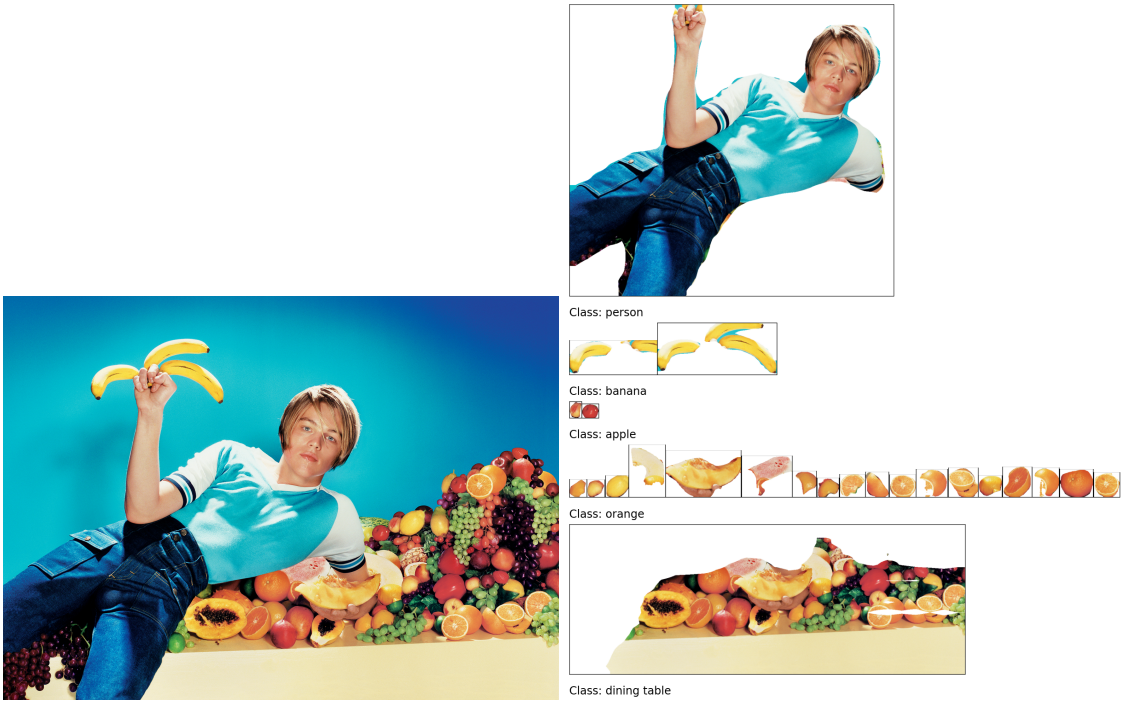

Figure 5.4: Sample input and output image

5.6 Building the application

After completing the tidied up image, the next step was to build an application that can load an image, run the model in inference mode and return the tidied up image from the input image. A difficulty was that MMDetection, like most deep learning frameworks, needs a Nvidia GPU to run even in inference mode. To solve this problem, a wrapper called SMD was used, that takes in MMDetection models and can run the inference on an arbitrary CPU. [2]

To convert the program into a full-blown web application, a framework, called Plotly-Dash has been used. Plotly-Dash is using Flask to create a webserver and is using HTML-, CSS- and JavaScript-technologies under the hood to create a running web application from a Python program. It offers out-of-the-box user interface (UI) components, that are useful to rapid prototype a web application. With the help of UI-elements like buttons and dropdown-lists, the user is given the possibility to upload and select images and models and to start the inference by himself. There are also two sliders built in: One to adjust the confidence threshold score of the model and one to adjust the input image size. Executed time was measured and gets written too to get an insight into performance of the model.

One of the goals of the web application was to build it as modular as possible. This was achieved with the possible selection of the model and three ways to select an image from (predefined list, upload and retrieve via URL). One difficulty with Plotly-Dash was that images when uploaded to the web server, got encoded with base64 encoding. It took some

time to find out, how to decode them for further use. A screenshot showing the web application and its UI can be seen here:

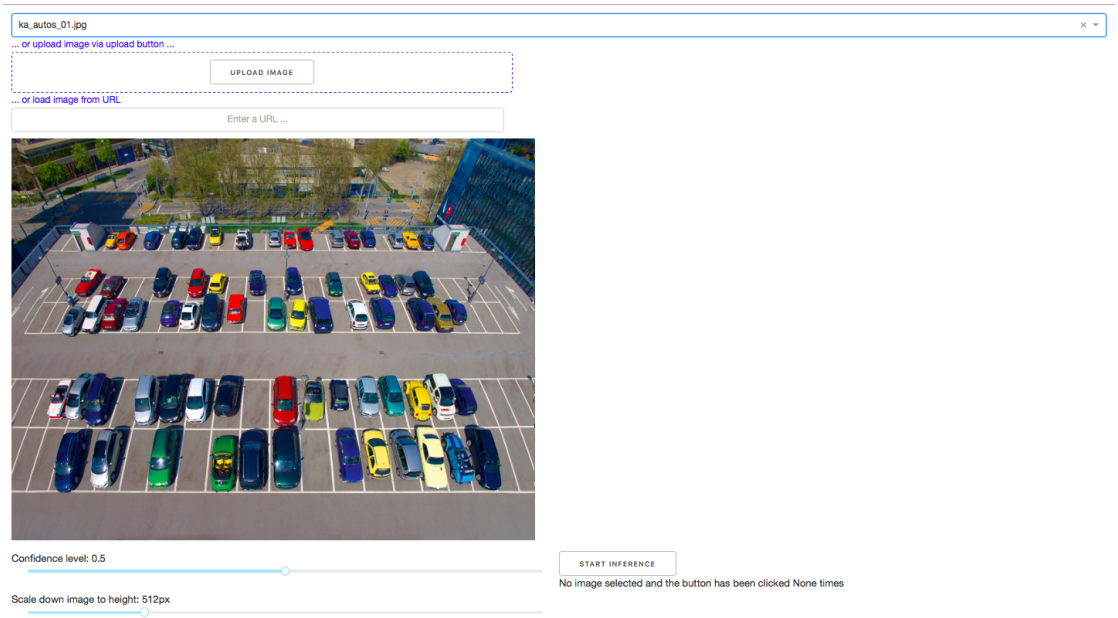

Figure 5.5: Screenshot showing the web application

5.7 Deployment

To deploy the application, server space on the EnterpriseLab has been allocated. The web application can be accessed at: `http://bdaf20-iameyer.enterpriselab.ch`.

5.8 A more precise research question

As Urs Wehrli luckily gave us his confirmation to use his photos from the series "Kunst aufräumen", we decided to use these images for retraining one or more models. Each model gets trained on the tidy version of an image and gets tested on the messy version of it. Optimally, the model should find all objects that are visible in the messy image and classify them correctly. There are some difficulties though. Per example in some pictures are most objects hidden because they are obscured by other objects in the foreground, as can be seen in the following picture:

Figure 5.6: Most of the objects on the image are obscured by other objectss

Optimally, the output of the model should contain exactly the same objects which are shown in the tidy version. For some photographies in his series, Urs Wehrli does use a particular technique to sort objects further in the "tidy" image. Per example using the colour or the size of the found objects.

5.9 Data labelling

A dataset containing of three images from the series "Kunst aufräumen" by Urs Wehrli has been chosen to refine the model. As each of these three images contains a messy and a tidy version, it does make a good template to use the tidied up version as a training image and the messy version as a test image. To accomplish training with own images, one has to label these images first. When training an object segmentation model, it is necessary to train the classifier with images that contain masks in it. There are several tools that offer object masking with polygons or brushes. A total of six images (three messy ones and three tidy

ones) have been labelled in order to use for refining the model. During labelling, a .xml-file gets generated, that after completion can be converted into the COCO-format as a .json-file.

5.10 Refining the model

Google Colab was used again as the platform to retrain the model. With just three training images it is quite extreme to train a classifier. This reflected in the poor performance in the beginning. After deciding to retrain a single model for every picture pair (messy and tidy one), performance got better. For a task like this, when training with a small dataset, data-augmentation is crucial: It lets the model use the same image over and over again after applying different kind of transformations and distortions to it.

When retraining with MMDetection, one has to adjust several things: A .json-file, containing all the classes, masks and bounding boxes in COCO-format. And a config.py-file that contains all the needed settings for the training process. During retraining, checkpoint-files get saved in a defined interval. These checkpoint-files can be used resume training on a later point of time. After one training cycle (between 1000 and 2000 epochs), the last checkpoint-file was taken and examined on the test-image.

5.11 Results

Hello

6 Evaluation and validation

6.1 Vergleich mit Anforderungen

6.2 Technische Aspekte

6.3 Vorgehen

7 Outlook

7.1 Conclusion

Hello

7.2 Outlook

Hello

List of Figures

List of Tables

Formelverzeichnis

Bibliography

[1] Girshick, R.B., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conference on Computer Vision and Pattern Recognition, 580-587.

[2] Karaniewicz, A. (n.d.). SMD (simple mmdetection). Retrieved May 30, 2020, from https://github.com/akarazniewicz/smd