Development of an Augmented Reality Engine Based on Real-time Object Detection

Zsolt Pasztori
Universitat Jaume I
Castellón de la Plana, Spain
zspasztori@gmail.com

Abstract

Augmented Reality applications let users interact with the physical world, while providing additional information from virtual sources. To overlay this information to real world items computers need to gain understanding of the video feed. With the recent advancement in computer vision real-time object detection became possible. During this project I have collected a dataset of a specific item. Carried out a survey of available object detection methods, focusing on their speed. Trained an object detector based on the YOLOv2 model with the custom dataset. Enhanced its performance during video feeds by using anomaly detection. Applied the object detector in a speech based real-time video editor application. The video feed is extended by broadcasting images, subtitles or translation on a white board held by the user.

Index Terms: Augmented reality, Object Detection, Deep Learning, Speech Recognition

1 Introduction

Before the age of computers information was available mostly in physical form. This started changing with the advance of computers, which was accelerated by the spread of Internet and smartphones. Today people spend a great amount of time in the virtual word. With the help of augmented reality, the advantages of virtual and physical domains can be melded. Augmented reality applications can revolutionize education, communication and media. With the recent breakthroughs in computer vision a better understanding of the environment became possible for computers. Using these methods I have decided to create an augmented reality platform.

Augmented Reality (AR) has been an interesting research topic since the 1980s. AR applications need to be able to understand the video scenes for positioning virtual objects on them. My current work focuses mainly on the computer vision side of the problem. Before the advance of deep learning feature based object detectors were used for object detection. The most widespreadly used ones were Haar-cascades [1]. As neural networks experienced a resurgence computer vision models started to be based on convolutional layers [2]. With the help of convolutional layers, and powerful regularization and training methods high accuracy image recognition became possible. Objects can be detected by applying image recognition to image patches. These patches can be sampled randomly, with sliding window, or by using a separate window proposition neural network [3]. Later specialized end-to-end object detectors emerged based on deep learning. For real-time applications object detectors optimized for speed were needed. The Tensorflow Object Detection API [4] is a collection of the most mature algorithms. The DarkNet framework was created for fast computation, it contains the YOLOv2 model [5], which is state of the art (SOTA) both for speed and accuracy.

Based on the DarkNet framework I have developed an augmented reality engine. The engine uses a model trained on the YOLOv2 architecture for detecting white boards on the images. I have collected and labeled a dataset of images.With the dataset trained a neural network. To enhance the performance of the network I have implemented a specialized filter. The final application uses the user voice input for user interface. The video feed of the real world is augmented with subtitles of speech, their translation or images depending on the user's needs.

The paper is organized as follows. Section II contains the conceptual design of the system, and collection and preprocessing of a dataset. Section III defines the problem from a computer vision view, recounts the previous work done in object detection, then explains the training of a YOLOv2 object detector. Section IV goes trough the application architecture, emphasizing the aspects of speech recognition, and filtering of the output of the object detector.

2 Experimental Setup

I could not find any open-source software to base my augmented reality engine on. To be able to evaluate the development first I had to make a system description. Every machine learning system has two equally important parts: data and algorithm. Since my use case was unique, I could not utilize already existing datasets. I had to first collect and preprocess my own data.

2.1 Conceptual Description of the System

For augmented reality to be useful for the user, the engine must provide real time video feed. In my case the main sys-

tem goal was to run with about 24 frame per second. This means that one cycle of execution must take less then 50 ms. The video input and video output needs lot of resources. These methods are highly optimized for speed and memory capacity. For this reason I had to focus on the object detection algorithms optimization. It is also important for the user experience to have no missing frames. Another important point is to have small movement of the virtual elements. These last two points are needed to maintain fluidity between the video frames. Finally, the time delay must be constant and small, so the user can interact with the environment.

2.2 Data Gathering

I have examined all the major computer vision datasets, such as ImageNet Detection [6], Pascal VOC [7] and Microsoft COCO [8]. Unfortunately, these datasets contain a limited number of object classes. They mainly focus on automated driving, pets and a few household objects.The augmented reality engine was going to be run on a laptop, for this reason the application had to focus on objects which could be found indoors. I had to collect my own data.
In my project I have decided to use the simplest dataset possible. I have decided to detect a single object category in the picture. This does not take away from the extendability of the system, since the neural network used for a 1-class and n-class problem has marginal speed difference. My object of choice was a white paper, which can be easily held in front of a camera, or used as a whiteboard. As mentioned before, no similar object can be found in already existing datasets.
First I needed to collect images, where there is a whiteboard or a person holding a paper. I had to find a source with hundreds of images, because collecting images one-by-one from image storage websites, such as tumbler.com or picasa.com, would take too much time. Reddit.com is a website which has several subforums dedicated to narrow topics. One of these topics, so called subreddit, is people taking pictures of themselves and asking strangers to make fun of them. The subreddit can be accessed at https://www.reddit.com/r/RoastMe/. For authentication purposes users need to hold a paper in front of them containing their name. Most of these pictures were good candidates for my datasets, since they contained the white paper as object. They were taken indoors, with similar light conditions and environment as my use case.
Reddit.com exposes an API for public use. The API has python bindings, and allows developers after registration to access the content of the website. I have written a python script to go trough the topics in the forum, and download the available pictures. Some of the photos are not hosted on the reddit.com website, rather on specialized websites, such as imgur.com. I had to write some additional code to retrieve those images too. During the download pictures are converted to jpeg format for better handling. This way I have obtained around 2000 raw pictures.

2.3 Preprocessing

The pictures obtained in the previous step had several problems which needed fixing. I went trough all the pictures, and discarded the ones which had some major faults. They could

Figure 1: Images from the dataset

be too low resolution, they might not contain the object, or the object was too small to be used for training. With this first round I had to eliminate 60% of the dataset.
Since the machine learning task was classification I needed to assign labels and bounding boxes for the objects present on the picture. There are several software developed for image labeling. I have decided to use Sloth bounding box tool, because it was the easiest to install, and had a simple interface. The labels and bounding box coordinates are stored in a separate json file. Unfortunately hand labeling pictures is a really time consuming task. I have spent around 20 hours to label the dataset.
After the labeling I wrote a python script for loading the dataset into arrays. The script resizes the images into standard sizes. It also converts the bounding box coordinates into several formats. This was needed to be able to experiment with different algorithms. Most algorithms use the simple top-left and bottom-right corner for the bounding box. Others may use one corner and the center of the bounding box. Some frameworks require the absolute pixel values of the coordinates, while others use relative coordinates.

3 Video Frame Processing

3.1 Problem Definition

An augmented reality application has to gain some understanding of the video feed. There are two main ways to achieve this: object detection and instance segmentation [9]. The approach taken influences the way the dataset has to be processed. Their decision impacts the problem formulation.
During object detection in the image objects are located, and bounding boxes are assigned to each object. The algorithm returns usually two coordinates from the bounding boxes, this means that the output bounding box will have parallel sides with the image. From object detection it is not possible to evaluate the orientation of the object. Object detection hence returns the location of the object, and its approximate size.
Instance segmentation task is to return a label on the pixel level. It finds objects and creates a mask, where every pixel which belongs to the object is labeled. It gives a much better understanding of the scene. There are several algorithms which consider instance segmentation as a further step of object detection, such as mask R-CNN [10]. Instance segmentation gives better results compared to object detection. The reason why I chose to use object detection is computational

considerations. Instance segmentation needs more data to train. It requires pixel level labeling of the dataset, which is much more time consuming. The training and evaluation time of instance segmentation is also higher. It must be noted tough, that the algorithms available for object detection are much more mature, and in the near future instance segmentation might become faster. The same phenomenon happened to the detection with the advance of R-CNN [11], fast R-CNN [12] and faster R-CNN [13], where each successive algorithm achieved a hundred times speedup.

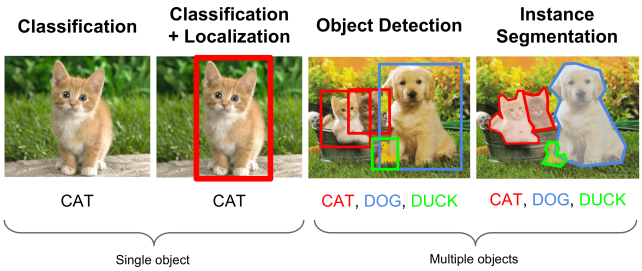

Figure 2: Difference between computer vision tasks

3.2 Object Detection Methods

Computer vision went trough a revolution since the 2012 ImageNet competition. Researchers realized that deep neural networks built on top of convolutional networks are more capable at this task then any previous method. These changes have first effected image recognition task, in which today computers have achieved the same accuracy as humans. After image recognition a much harder problem came into focus. The problem of object detection, where each network is based on a simpler image recognition network. Most algorithms historically focused on maximizing the algorithm's accuracy. These bulky models proved to be too slow for actual application. Recently, more and more new frameworks and models are optimized for speed. In an augmented reality application a lower precision is enough, speed is the main concern. To be able to choose the best model for my use case I have coded and trained 6 different type of models.

The object detection model which was used widespreadly in applications before neural networks were Haar-cascades. Haar-cascades are feature detectors. They are an ensemble of linear detectors. They provide high speed. Most hand-held cameras and phones use these for face detection on images. They can achieve real-time detection even on embedded devices. Their drawback is sensitivity to lightning and diversity of the training dataset. They are extremely hard to train, and the training depends on the complexity of the objects too. The more complex the object, for example a face, a wrist watch or a car, the easier it is to train the algorithm. On the other hand for objects without too many features, in my case a white board, are unfeasible to train. I have attempted to train Haar-cascade contained in the OpenCV framework. Regardless of the hyperparameters the model did not converge.

Convolutional networks are type of neural networks. They take into account the spatial ordering of information. In an image each pixel is strongly correlated with the pixels close to it, and less as the distance grows between them. This realization was the step which allowed to move away from multilayer perceptrons in computer vision. Dropout and maxpooling are methods for regularizing neural networks containing convolutional layers. The usual image recognition systems contains blocks of these, and a few densely connected layers at the bottom. Two convolutional layer, followed by one dropout and one maxpooling layer consists of one block. I have experimented with using this kind of four layer block architecture for building an object detector. I had to realize that although maxpool and dropout are needed for regularization, they result in the loss of location information in the network. This makes it not possible to train a similar few level network for object detection.

The main difference between image recognition and object detection is the layout of the input image. In image recognition task, there is a single dominant object on the picture, such as a cat or dog. In object detection there might be several different objects on the picture. Object detection can be considered a generalization of image recognition. For image recognition current algorithms are extremely fast, and their precision is on a human level. Several object detection algorithms are based on the idea to apply image recognition to patches of a picture. After applying it to patches, several objects will be found, and the algorithm just chooses the patches with the highest confidence. These patches finally become the predicted bounding boxes. Models built in this way are called two-stage object detectors.

The main difference between the two-stage methods is the way they sample the bounding boxes. The easiest way is to shift a rectangle along the axis, and change the size of the rectangle. In this way smaller and bigger objects can be found regardless of location. This is called sliding window method. The main drawback of this method is its execution time. For each patch the image recognition network has to be used. The number of patches can be in the hundreds, which make it infeasible for real-time application. I have implemented a detector in the Keras [14] framework. The main problem for my implementation was not only its speed, but also because of the random sampling of the patches the box locations were inaccurate.

Another two-stage method is based on windows proposition method. Here the patches are not selected random. They are chosen by another neural network. Only parts of the image are evaluated which have a certain diversity and unity. This means that places which are uniform of color, or are surrounded by edges are more likely to contain objects. I have found two main drawbacks for this system. First, the window proposition network is not feasible to be trained for custom data. It would need a separate training dataset, where not objects but also patches must be labeled. The other drawback is that its speed is still not good enough for real-time usage. In the Tensorflow framework to optimize for speed it is possible to evaluate several pictures in batches. Evaluating batches is fast, but the window proposition network can not use this trick for speeding up. I could achieve around 1 FPS with it.

During June of 2017 the Tensorflow Object Detection Api was released. It is a collection of object detection algorithms

built on top of the Tensorflow framework. It contains several two-stage detectors, such as faster-RCNN based on ResNet [15], MobileNets [16] and VGG [17], and also Single Shot Detector [18]. The Object Detection API was developed by taking speed/accuracy trade offs in mind. The big advantage of this framework is that with small amount of boilerplate code it is possible to compare different algorithms and evaluate their accuracy and speed. I have written a full system based on it. The system took live feed from a webcam, then used the object detectors to evaluate the pictures. The accuracy was satisfying and speed was promising. I have used several tricks to speed up execution, such as asynchronous threads, multiprocess communication. Unfortunately, I could not achieve a higher speed then 7 FPS. I had to realize that there is a bottleneck in the Tensorflow framework. The bottleneck was between the transfer of the images from the RAM to the GPU. Regardless of algorithm it was not possible to reduce the inference on a single picture under 50 ms. I did not want to spend time on trying to correct an architectural problem, so I have gave up on it.
Finally, I have came across the YOLO object detector. It is optimized for real-time usage. It has comparable precision to the more robust models. This was the only detector I was able to run in real-time on my hardware. I have used this for my final implementation.

3.3 DarkNet Framework

DarkNet is a deep learning framework written in C++ and CUDA. It was created, and is maintained by a single person. For this reason it is minimalistic in design, hence it strives for simplicity. Its main drawback is its low user count. It is exceedingly hard to find any information on how to solve problems. It has no documentation. It contains several known bugs. Since it is written in C++ it is not possible to use it directly in Python. Python is today the de facto language of machine learning. Its advantage is fast prototyping and dynamic access to memory. These features are lacking in DarkNet, which makes development in it slow and painstakingly prone to bugs. DarkNet lacks any kind of meaningful application development interface. Even in C++ it is not possible to change parameters in execution time. Every parameter has to be changed in code, and later the framework must be recompiled. This is annoying, considering that training a model, inference on the model, and using it in live application requires different modified and compiled versions of the same framework. DarkNet even tough its several limitations is used by a number of researchers. The reason for this is the YOLO architecture. The You Only Look Once object detector was released to this framework, and as of today it is still its fastest implementation. This can be attributed to the fact, that most high complexity deep learning frameworks are optimized for inference on batches of images, and feeding images from memory to the GPU one-by-one is slow. This is the case with TensorFlow and Theano. DarkNet has no issues with transferring individual samples between GPU and RAM.
The YOLO architecture was created with keeping speed in mind. It divides the picture into a lattice of equal size rectangles. When detecting an object it detects it for these rectangles individually and later unifies them. This way inference is several times faster compared to faster R-CNN, which in theory achieves better precision for bounding boxes. Despite YOLO architecture's theoretical limitations, its accuracy is better compared to faster R-CNN and SSD. This can be attributed to the much more sophisticated training methods for the system. The original pretrained model, uses several tricks to enhance performance. These tricks include: data augmentation, batch normalization, and multiscale prediction. In contrast to other methods the author here does not use purely supervised classification. During training the software builds a word tree based on the WordNet model, and uses the whole ImageNet dataset, augmented with Pascal VOC and Microsoft COCO at the same time.
There are two main versions of the YOLO network. The original version has a precision comparable to SOTA methods. This model executes around 10-14 FPS on my GTX 950m videocard. There is also a lower precision, but faster model. This is called tiny-YOLO. It has fewer layers, but can execute it with 20-24 FPS. I have chosen tiny-YOLO for my implementation, because its accuracy was acceptable for the given use case.

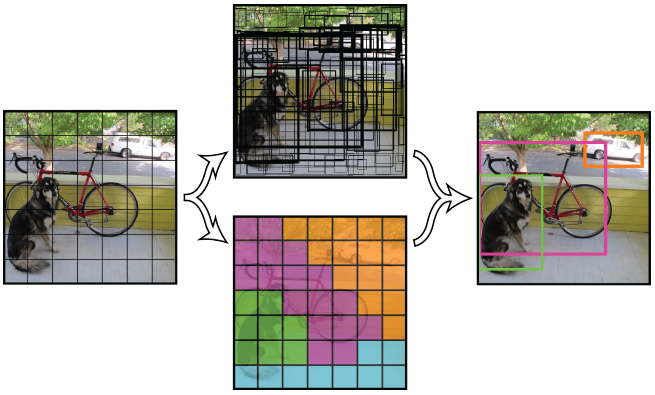

Figure 3: Yolo model

3.4 Model Training

Object detection models are infamously hard to train. Most models can not be trained end-to-end, because of their sensitivity to initial weights, the first set of training batches, and hyper parameter configuration. High-complexity models need lots of data and time to converge even in optimal setting. Fortunately the author of YOLO has shared a pretrained model for both YOLO and tiny-YOLO. This pretrained model was trained on the ImageNet 1000 dataset. It contains 1000 different object classes, and enough diversity between the classes to be a good starting model for training. The existence of pretrained model was crucial for me. I only had a dataset of around 800 images. This would not have been enough for the model to converge from randomly initialized weights.
I have first trained the initial model on my own computer. Staring from the pretrained network it took 12 hours for the model to converge. Unfortunately the results were poor. The

precision was promising, but the system had low confidence and big false positive count. I had realized that my dataset is too diverse, and the bounding boxes are not fitted correctly. I had to discard another 500 images from the already small dataset, and only kept the best and clearest examples. On the other hand, I had to relabel the remaining pictures by hand. This time the bounding boxes would contain every time the whole paper, and a small part of its surrounding.
My final dataset consisted of just 250 pictures for training, and 30 pictures for validation. DarkNet has built in image data generators for training. They automatically generate new pictures based on the dataset. This pictures are created by resizing, translating, rotating and changing the pictures contrast. I have experimented with using the initial dataset for pretraining the model, and later doing another training round with the higher precision dataset. The dataset of 250 training pictures proved to be enough for training the model, with the help of data augmentation and pretrained model.
I have trained around 25 different models, and chose the best performing one for the application. Each training took around 12 hours. The main difference between them are hyper parameters, and pretraining. I had to use Google Cloud machines, equipped with Nvidia K80 GPUs to carry out the training. During training I have monitored the validation and training losses to asses training. The training loss value have decreased from thousands to around 0.5 with time. I did not log precision and loss values, because they did not correspond well with how smoothly the model run during prediction. I had to evaluate each model with real movements of the object in front of the camera. The loss values were only used for deciding model convergence. For each model the full convergence took around 10000 passes over the dataset. After 5000 passes the bounding boxes started to converge, but the confidence of classification was low, and reacted poorly for unusual orientations. The final model was trained for 13000 rounds. Both the precision and confidence for this model is high. After 14000 epochs unfortunately the model starts to overfit the training data and during detection the position of training image boxes can clearly be seen.

4 Application

My main goal with this project was to make a survey of computer vision algorithms. I have concentrated on the object detection task because it is at the forefront of research today. At the same time it is mature enough to be used in a production environment. In my later works I wish to use the obtained knowledge in robotics. Because of hardware limitations and the inherent difficulty of developing robotic applications I had to find a different application to show its capabilities. For this reason I decided to implement an augmented reality engine. Unlike robotic applications, both the input and the algorithm's output can be judged by the user.

4.1 Application Concept

Most augmented reality applications currently available are focusing on entertainment. Probably the most famous of these applications is Pokemon GO. I on the other hand wanted to make a framework for making videos in real-time.

Figure 4: The detector in work

These videos can be later uploaded to websites such as facebook.com or youtube.com . The usual method of creating video content involves making several takes, then applying computer effects on it, finally cutting the video together. This method is troublesome. The video creators usually need several days to process and cut a video together. I wanted to speed this process up by allowing users to make a single take, apply the visual effects real time, this way completely bypassing the stage of video editing. The user can activate visual effects by the use of keywords. During the video take the software looks for speech queues and reacts to them. For example if the user says the keyword "Uruguay", a map of Uruguay will appear on the whiteboard behind the user. The visual effects I chose to showcase are subtitles and pictures. Later videos and even virtual visual interfaces, such as buttons and knobs, can be added. These elements are broadcasted on the paper held by the user. The object detection module is responsible for finding it on the video feed.

4.2 Software Architecture

Making an application with the intention of creating a real-time user experience is challenging on the software side. My application contains audio and visual input, with visual output. It is not possible to run them sequentially, since audio and video input arrives parallel from the user. The software had to be inherently parallelized and every thread had to be made asynchronous, to avoid blocking the pipelines, meanwhile reducing the response times. The final program contains 5 pipes and 7 parallel threads of execution, it uses one on-board neural network, and 2 additional neural networks through cloud APIs. The program uses both the processor and an Nvidia GTX 950m video card for computation. The software is mainly written in Python 2, but it calls the DarkNet C++ framework trough a python wrapper.
The processes can be separated roughly into 3 pipelines: vision, speech, and output pipeline. The vision pipeline is responsible for obtaining the camera images, and finding the bounding box of the white board. The images are acquired

from the computer webcam with the help of OpenCV. The images are passed to the DarkNet python wrapper, which passes it trough the network and returns the bounding box coordinates, if an object is present on the image. The bounding box coordinates get filtered, and finally returned to the output pipeline.

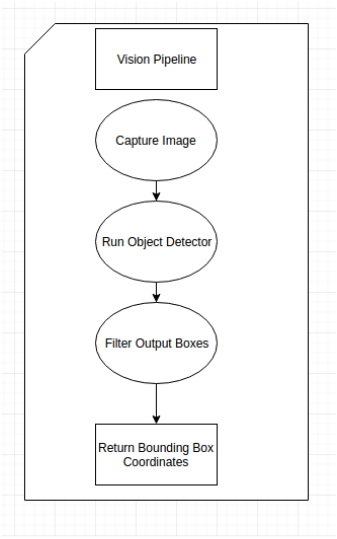

Figure 5: The vision pipeline

The sound pipeline is responsible for capturing audio and transforming it to text. I have developed this functionality during a previous project. The audio is captured with Portaudio. The stream is broken into 50 milliseconds chunks and streamed to a server. At the server the Google Speech API passes it trough a neural network, which returns completed sentences. The speech language of the user must be specified beforehand, but it can be modified during execution by the usage of sound commands. The streaming speech recognition is almost real-time, there is around half a second delay between the end of the sentence and the arrival of the transcription.

The output pipeline is responsible for creating the images, which the video feed contains. From the result of the speech transcription the Natural Language Understanding module decides what action to take. It draws an image on the paper if a predefined keyword is in the text. Otherwise it prints the subtitles of the picture. The subtitles can be the original text, or it can be a translation. For example if the user says ”Please translate from Polish to Catalan”, it means that the audio input language will be changed to Polish, and the subtitles will be translated to Catalan. The automatic translation is carried out by the Google Cloud Translation API. After the text or image is drawn on the picture it is shown to the user.

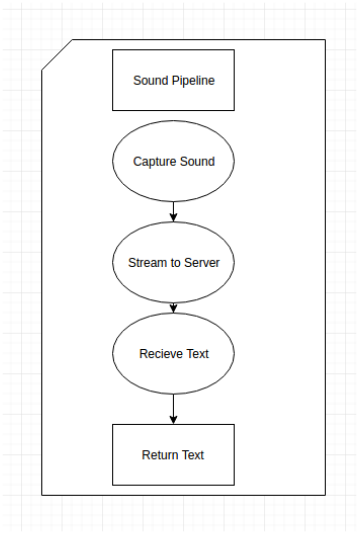

Figure 6: The sound pipeline

It is important to notice that that the output pipeline is asynchronous and its timing is independent from the vision and voice pipelines. Every frame from the camera input is returned as an output with a constant delay. Although here is a variable delay between the effect of the speech commands.

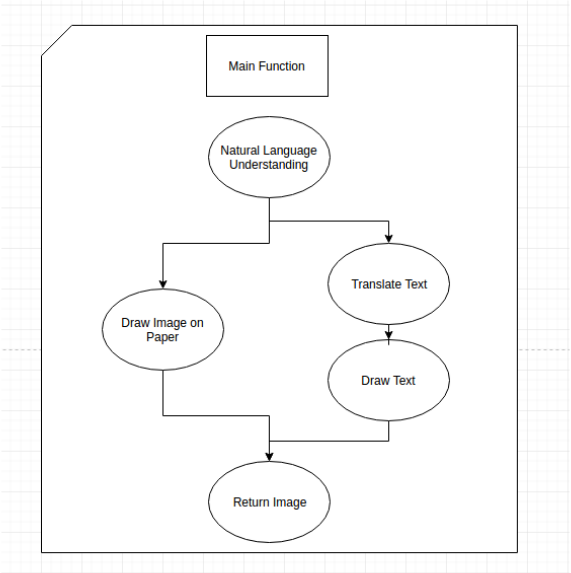

Figure 7: The output pipeline

4.3 Temporal Aspects

Object detection as a topic aims to label and locate as many objects as possible on an image. An augmented reality application on the other hand uses a set of images, a video feed. There have been recent developments into segmentation in video feeds, such as the Clockwork Network [19], but these are only initial methods. In a video feed every image is highly correlated with the previous image. In the future I expect the emergence of neural networks which are able to base their detection predictions on not just current images, but also previous outputs of the network. This will clearly boost their performance. By using tiny-Yolo the framework is capable of processing every image in real-time, even tough this might not be computationally optimal. The problem which arises in this case comes from movement. If the detected object moves the bounding box coordinates will move too. At some frames the object might not be detected. If the orientation of the object changes, the bounding boxes might move in an unnatural way. This flickering may ruin the user experience.

To discard the flickering effect I have decided to implement a filter, which calculates the bounding box coordinates based on previous coordinate locations. I have experimented with using a simple low pass filter, median filter, and band-pass filters. Later I realized that the flickering effect comes from single defected points. To filter these out I turned to anomaly detection. I have implemented the TEDA framework [20]. The framework is based on the assumption that most points of the time series are typical. The points should be compared with each point of the time series instead of the average point of the series. This way the outliers can be found easily, without depending on a metric, which is influenced by them. Since the average is influenced heavily by outliers. Even if the series goes trough a fast shift the previously thought outliers are not discarded, but takeover after a few time steps. This method allowed the augmented reality output to be much more fluent, leading to a better user experience.

4.4 Speech Recognition and Translation

The speech recognition task built into the software framework is called Text-to-Speech (TTS) in the literature. There are three main ways to perform TTS: on the computer trough feature based methods, on computer with neural networks, and trough a cloud provider. I have tested all currently available methods and decided to use Google Cloud Speech API. Cloud Speech API had the best performance given my non-native English accent. It performs well regardless of sampling rate. It works relatively well in noise environment, allowing to be used well from even a webcam microphone. Another helpful feature is that keywords can be provided for it, which the network chooses even from lower confidence. The keywords for the speech commands can be provided as keys at speech recognition, ensuring that user commands are detected with higher chance.

Almost all available speech recognition software can be used only in synchronous mode. This means that the voice files are only transmitted to the server when a sentence is finished by the user. During the recording of the audio sentences must be recognized, which can lead to errors from bad segmentation of sentences. On the other hand it increases the delay between the audio and the text output, since the whole sentence must be uploaded at once after the sentence has ended. Google Speech API gives an option for streaming recognition. During streaming recognition packets of data are constantly transfered to the server. The server returns the text response, when it detects a full sentence. With this trick the recognition task delay can be reduced to half a second, compared to the usual 8-12 second delay for synchronous methods. The streaming recognition feature is still in beta, and no template application is provided by Google. I had to write my own implementation. The implementation itself contains 3 asynchronous threads for the best performance possible. It exploits heavily the lambda function concept available in Python.

For text translation I have used the Google Cloud Translation API. It can translate between hundreds of language pairs. It can be accessed free of charge for low volume use. A python API is available for it, made by Google. In my experience it is quite reliable, and fast.

Figure 8: A frame during English-Italian translation

5 Discussion

During the period of 3 months I have created an augmented reality application based on object detection. Collected and labeled a dataset of images containing people using papers as white boards. I have performed a survey of deep learning methods for computer vision. Coded and evaluated a series of object detection algorithms. I have found out, that currently in speed, accuracy, and ease of training the YOLO model in the DarkNet framework was the leader. Trained a model in the framework using a pretrained model on ImageNet and the custom dataset. To counteract the drawback of the neural network using only the most recent image I have implemented the TEDA anomaly detection framework. Used the model to build a multi-threaded asynchronous application. The application uses speech recognition and machine translation for the users convenience. It can broadcast images and subtitles on a white board or paper. The user can direct the application

with voice keywords.
For future research it would be interesting to experiment to extend the object detector with 3D convolution or recurrent neural network layers. This way it could be possible for the system to make its prediction also on previous images. Another direction would be to step over object detection and work on real-time instance segmentation. This would give a much better result, since the object's orientation and more precise position could be found.
The augmented reality engine is successfully working, but it could be later extended to fulfill more user needs. Currently it can only detect one type of object, with a better dataset this could be changed to detect several objects. Additional effects could be added to videos, such as moving animation and video broadcasting on objects. Instead of simple speech commands a better natural language understanding environment, such as api.ai or wit.ai, could be used. Instead of using the object detection only a way to position the output virtual visual user face could be created. Button, plugs and sliders could be broadcasted on the white board, and could be modified by the user. This way the augmented reality could serve not only as a way to output information, but also as a visual user interface.
A video of the application can be found on the following link: https://www.youtube.com/watch?v=RY7PWVt2vWE .

References

[1] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Image Processing. 2002. Proceedings. 2002 International Conference on*, vol. 1, pp. I–I, IEEE, 2002.

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

[3] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International journal of computer vision*, vol. 59, no. 2, pp. 167–181, 2004.

[4] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," *CoRR*, vol. abs/1611.10012, 2016.

[5] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results." http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[8] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[9] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. G. Rodríguez, "A review on deep learning techniques applied to semantic segmentation," *CoRR*, vol. abs/1704.06857, 2017.

[10] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[11] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013.

[12] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015.

[13] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[14] F. Chollet *et al.*, "Keras." https://github.com/keras-team/keras, 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

[19] E. Shelhamer, K. Rakelly, J. Hoffman, and T. Darrell, "Clockwork convnets for video semantic segmentation," in *Computer Vision–ECCV 2016 Workshops*, pp. 852–868, Springer, 2016.

[20] P. Angelov, "Anomaly detection based on eccentricity analysis," in *Evolving and Autonomous Learning Systems (EALS), 2014 IEEE Symposium on*, pp. 1–8, IEEE, 2014.