EWI3615TU - Computer Science Project

Using Computer Vision to search YouTube Videos

Group 1

Gilbert Hardeman	4357337	Daniek Dieben	4448138
Lisa-Marie Plag	4969669	Britt Rooijakkers	4551451
Sean van der Meer	4445546		

Delft University of Technology

January 25, 2019

Delft University of Technology

Challenge the future

Contents

1 Introduction

Computer vision is one of the most popular applications of artificial intelligence techniques that there is today. It helps simplify everyday life in many ways by enabling machines to recognize objects in a certain environment. Methods such as image recognition can be used instead of a text based search, making it faster and more convenient to find media related to a given topic. One source of media that is consumed regularly by most people nowadays is YouTube videos. Consequently, the idea of this project is to make YouTube searches easier and more efficient for its users by adding a tool to search videos based on a picture. This way, a user who wants to find a YouTube video containing an object that is in his or her environment can simply take a picture using his smartphone camera and immediately obtain the results without having to type any text.

Figure 1: Image recognition

2 Development Plan

2.1 Project Objectives

The main objective is to design a system that allows for an image based YouTube search by recognizing an object in a given image, extracting relevant search terms and collecting the IDs of corresponding YouTube videos. Therefore, our project objectives can be split into two separate aims. First, we need a suitable neural network that can recognize objects in a given image and assign relevant keywords to it. Second, we need a way to search YouTube for videos containing the search terms that were extracted by the network. The user of the software can then decide which YouTube video he or she wants to watch on the topic given by the picture without having to provide any textual input.

2.2 Realization Plan

The object detection task will be addressed by training a neural network on a dataset of labeled images, such as the Common Objects in Context database (COCO) of Microsoft. This dataset contains 328,000 images including 2.5 million labeled instances of 91 object types. Examples of object categories include electronics, food, animals or vehicles. The plan is to decide on a couple of specific objects from these categories to train the neural network in image recognition. As it would be useful in practice to be able to take a picture of certain groceries and find a cooking video containing a certain ingredient quickly, we decided to pick three food products to focus on. Hence, our main aim is to recognize apples, bananas and broccoli.

Figure 2: Example image of vegetables from COCO database

Using the extracted search terms provided by the neural network to collect relevant YouTube video IDs was planned to be realized using the open source 'YouTube 8M' data base, containing 6 million video IDs and more than 2 billion audio and visual features. The data on videos cannot all be stored entirely on one single machine but should be processed and checked for the given search term immediately so that only relevant video IDs are stored. The YouTube component makes up the big data part of this project. However, we decided throughout the process of implementation that using web scraping with Beautiful Soup to retrieve video URLs from YouTube would be more feasible given the limited amount of time we had.

2.3 Project Progress

2.3.1 Week 1

During the first week, a lot of time was spent on initialization of the project. We got instructions on how the project was structured and we met the other team members. We started brainstorming right away about the objective of our project and what we wanted to achieve and what not. We decided not to use Scrum completely, but combine a few aspects of it in a way we see fit. That means we wanted to sort our goals on priorities as to have a structured progress of the project.

Deciding on our objectives was not easy. We struggled a bit with what was possible to do and what was challenging enough to make. In the end we settled on using computer vision to search for YouTube videos. This way we really combine the Computational Intelligence part and Big Data part of the project. Computer vision is used to recognize objects in images and the YouTube database is used to search for videos which contain this objects.

The main goal is to make a picture of some object (we had not specified yet which objects we want to train the neural network on) and let the YouTube database return video IDs of videos containing this object. We did not specify if we wanted to return one video, the best video, the video which contains the most of this object or something else. We found Microsoft's COCO database which we could use for training the neural network on image recognition and we found that the YouTube 8M database contains a lot of useful data and also some documentation on how to work with it.

The first priority is to design a neural network in such a way that it recognizes given objects in images and that it is able to give the correct name of the object in the picture as an output. Based on related literature, it seems most suitable to use a multilayer convolutional neural network (CNN) to recognize objects. Subsequently, the second priority is to develop a way to feed the object name to the YouTube Database where it searches for all or some of the video IDs containing this object in an automated fashion. In the end, we want to combine these two. Our last priority is optimization, which can still be done if there is time left at the end of the project schedule. As result optimization, the neural network can be extended to include functionalists like: filtering background noise in the images and applying error correction using unsupervised learning.

Also, we want to have plannings at the beginning of the every week as to measure our progress and divide all of the work over the team members evenly. We also planned to already write evaluations week by week in the report as to have a clear overview of our progress.

2.3.2 Week 2

In the second week, the development plan draft had to be finalized and a meeting with our supervisor Casper was scheduled. During the planning of this week we decided that our short term goal for this week was to finish the draft and ask Casper about the general idea of the project, what are the criteria for the report and the software implementation and where our focus should be.

We learned from the meeting that the project was mainly about structuring a large project, making good plans and following those and achieving the goal in a structured manner. We also needed to make our implementation goals more concrete, so we needed to specify which objects we want to recognize and how many video IDs we want to return from the YouTube database. Also, we need to specify if we want to use already existing modules to implement the neural network or not. Furthermore, the first four weeks of the project are meant for doing research and we should start implementing after that.

The rest of the week we did a lot of research on how to work with the YouTube 8M database and how to set up a neural network for image recognition in Python. For the first we found two interesting cluster computing frameworks, Spark and Hadoop. We figured that Spark was also a part of our Big Data Processing course, so we decided to look into that one instead of Hadoop. Also, we found that Spark is 100 times faster in memory than Hadoop (`https://www.quora.com/What-is-the-difference-between-Hadoop-and-Spark`).

To set up a neural network for image recognition, we found TensorFlow, which is an open source machine learning framework by Google. The most popular high-level API module of TensorFlow is Keras. However, we decided not to use this module as we found this was too high-level and wanted to do the implementation of the network ourselves using only TensorFlow.

We already had found Microsoft's COCO database. We also found ImageNet Database, but this required authentication, so we decided to use COCO database for training the neural network on image recognition. We settled on training the neural network on recognizing apples, bananas and broccoli. We settled on food objects, because we saw usefulness in obtaining YouTube videos about food. You can for example make a picture of three leftover ingredients and obtain a YouTube video where all three are combined in a recipe video. If there is time left at the end of the project, we will try to make the CNN recognize multiple objects in one picture, but as a start we will try to make this work with only one food object in a picture.

2.3.3 Week 3

This week was meant for more research and deciding on our design plan, methods and tools. As a backup plan, we chose to use web scraping on the original YouTube website to process request calls. This can be done using the python package Beautiful Soup. The key returned by the neural network would then simply be used as a search term for the YouTube website to retrieve relevant links.

We also did more research on TensorFlow, watched videos on convolutional neural networks and followed a general beginner tutorial of the TensorFlow network. Also, we figured out how to load some of the data of the COCO database.

During the meeting with Casper, we figured out that it was a bit early in the project to already look at Spark and TensorFlow and interfaces in general. First, we should figure out how our data pipeline is going to look like and which format the data has. We should define a process which can pull data, process it and feed it into the framework. Also, we should think about our software design and architecture.

2.3.4 Week 4

In the beginning of week 4, we planned to do more research on how the data is going to look like. The images are plain *.jpg* files and the annotations are in *.json* files. We want to convert *.jpg* to tensor using the *tf.image module*. This module has a lot of functionalities for preparing data, such as normalizing images, dimensionality reduction and it has methods for achieving a uniform aspect ratio and scaling. The CNN can produce $n + 1$ different classifications (keywords). By feeding in the image data, it then converts the image to a keyword. We decided on also focusing on being able to handle cases where none of the keywords have been found. Also, we want to have the CNN clarify it to the user if recognition was successful.

The data retrieved from the YouTube database filtering is expected to be a list of video IDs or links, so we need to filter or sort this huge amount of data for the user. By default, the data is displayed as it is without any additional filtering or sorting. We have decided to implement additional functionality for sorting by relevance, upload time, view count or rating as given by YouTube. It is also possible to filter based on upload time and duration.

During this week, we also created the software architecture diagram, which shows how our software is structured and how data flows through the network (see Figure 3). However, there was no meeting with Casper.

2.3.5 Week 5 and 6

In the last week before the Christmas Break we started implementing all of our ideas. The CNN was implemented by Lisa and Gilbert. Sean, Daniek and Britt worked together on the YouTube data retrieval. First, we had planned that Britt would also work on the CNN but we saw it was more fit to let one extra person work on the YouTube data retrieval as it was a bit behind. We managed to download a small sample of the YouTube 8M dataset. However, the downloaded data was in *.tfrecord* format and we didn't know right away how to work with this kind of data on the fly. In the mean time, we started to implement some web scraping.

We implemented three scripts: a request verification script that checks if we get a valid response, a conversion script that converts the URL response to HTML, XML, JSON, etc., and a module script. Currently, we use a module for YouTube itself. In this module we can prepare search terms in the right format for the YouTube search engine to use and create a valid URL. This module can easily be replaced by a module for the YouTube 8M database if we still decide to do that later. The valid URL is passed on and eventually all watchable hyperlinks are returned from the web page. These links are already filtered by the YouTube search engine on most watched videos, but we can create additional filters for that if we want to.

Moreover, Daniek created a graphical user interface (GUI) where the user can choose between making a picture with its webcam or choosing a picture from the computer gallery. What still needs to be done is to couple the CNN and the web scraping part so that a whole data pipeline is constructed and data flows from a picture as an input to a YouTube video as an output. This is scheduled for after the Christmas Break so we can work together on that during instruction hours.

For the CNN we set up a draft version of the code. All the part that we need that we need like the actual network and ways of feeding in the pictures and evaluating the performance. It compiles without error and now we can work on improving the different parts.

2.3.6 Week 7

After the Christmas Break, we looked back at the work we did before and during the break. The CNN is implemented but still too slow and there are a couple of problems with training the network. For the big data part, we succeeded in filtering the YouTube videos on relevance, rating, view count, upload time, upload date (today, this week, this month and this year) and duration can be filtered on long or short. The default filter is set to relevance. We are also able to filter on definition, if the video has subtitles and if the video is live or not.

The only limitation we have encountered is that the video duration can only be filtered on below four minutes and over 20 minutes and the program does not return any videos in between. Now, the videos that are returned are from all over the world, so some videos are also in Russian, Japanese or some other language. We are not sure if this is a benefit or not. The GUI is also working as expected. However, we are not yet able to open the webcam from the GUI. It is possible to open the computer gallery from it. What remains to be done is to implement all the functional code (from both the CNN as the big data part) into the GUI and turn it into a working application.

This week we plan to implement the functional code for the big data part into the GUI and make a drop-down function for choosing bananas, apples and broccoli as long as we don't have the well functioning CNN yet. This way we will be able to see if refactoring leads to a well functioning application where we can input filters and search terms and get the correct videos as an output.

So far the results of the CNN were not satisfactory. We figured that this might be because of noisy data. In some of our pictures the objects that should be recognize are really small. This week we worked on ways to filter out those pictures. We also experimented with different CNN architectures

2.3.7 Week 8

The last week before the final presentation we have to wrap-up the implementation of our program. The two parts of our project are still not coupled together, so we need to make some progress there.

The big data part is as good as done. There were a couple of things we had agreed last week to do before we met again. These things were making sure that the GUI was able to switch from frame to frame and showing the thumbnails including a hyperlink of the videos on the GUI, instead of printing the links in the console, which was a bit shady. We also struggled a long time with coupling the thumbnails to their corresponding links. Showing the thumbnails was not hard at all and we fixed that very fast, but for some reason every thumbnail was linked to the same one link. Eventually, we fixed the bug and made sure to show 15 clickable video thumbnails at every manual search. Now, when you click on a thumbnail, it will bring you to the video on YouTube.

We encountered some problems still. Some videos do not have any thumbnails. In that case YouTube just shows a snapshot of the video, but a gray image was shown in the GUI instead. We did not know a way to get this snapshot, so we decided on just removing all the videos that did not have any thumbnails. Now, only videos with existing thumbnails are shown at every search. We also noticed that the manual search was a lot slower than before. It now takes about 15 seconds for the program to return videos on our GUI. We think this is too long and are thinking of ideas to make the program more faster. Probably, we can make some improvement in the part where the request is verified. However, minimizing the timeout between connect and read did not result in a much faster program.

Not all the videos that were returned were recipe videos. Some of them where about challenges of people to eat for one hour straight or eating as much bananas as possible. Lastly, there were some videos of people cooking naked and we decided that this was not something we wanted to show to the whole classroom when demonstrating our program during the presentation. So we also got rid of all the 16+ videos, so to speak. Lastly, we made some unit tests and we also made sure that the program we now have is ready for coupling it to the CNN as to make sure this won't take much time, in case the CNN is not ready to use this week.

This week we spent a lot of time improving the CNN. As is described in the technical part of this report we needed to reevaluate our complex CNN architectures. Unfortunately we found that our data did not contain enough pictures of apples to include recognizing apples in our design.

The tweaking of the network was a time consuming process since changing the architecture often means finding new parameters like the learning rate that work best with the changed architecture. In the end the we had a much more stable performing network that was ready to get connected to the rest of the system.

2.3.8 Week 9

Over the weekend we coupled the CNN, web scraping and GUI all together. As expected, this did not take too much time as we already thought about how to pass data from one part to the other. However, there were some unexpected problems with Python versions which differed for the CNN and the web scraping part and some modules that could not be installed, but in the end everything worked as expected. We now also had time left to implement the webcam feature where the user can take a picture with his webcam of a banana or a broccoli and input this picture into our program.

The last week of the project we planned to work on the report, evaluation document and presentation. The implementation was already done, so we had a lot of time for preparing the finalization of the project. We discussed a lot about how to structure it all and in the end we settled on copying the structure of the evaluation document to the presentation, as we thought that the most important objectives of this project were reflection and evaluation.

As the implementation was already done at the beginning of the week, we had plenty of time to prepare the presentation, which we thought went very well. We also had enough time to finalize the report and evaluation document as much of the work was already done. The parts left to write where the technical implementation of our program, comparison between software design plan and actual software design (including diagrams) and finalize the evaluation document. We also wanted to make a sort of classes diagram which explained our software architecture. We divided these parts evenly over the group.

Looking back, we had a really fun time during this project. There was a really good ambiance in the group, especially at the end of the project. We all think that we made a really nice program and also did we divide the work load evenly over all the group members. Also, we evaluated our progress every week and this turned out to be very informative. So overall, we are really happy with the end results.

3 Design Plan

3.1 Design Strategy

The implementation and design strategy followed during our project should be iterative and incremental, as our techniques will become more advanced throughout the learning process and the plan might change over time. Furthermore, we will try to follow the agile principles because not every detail of the tool we want to develop is known up front. We might realize during the process of implementation that there are additional tasks to be completed to reach our final aim or that certain objectives are not feasible. In this case, we will modify our aims and design as we see fit.

The critical features for this project are:

- The resulting software has to categorize the given image in either of the four different classes (e.g. apples, bananas, broccoli or none) and provide corresponding video links
- Every type of image format has to be accepted by the software and needs to be converted to *.jpg* with which the software can work.
- The image has to go through a preparation process; *noise-cancellation* and *error-correction*

- The image will be labeled by the convolutional neural network with one of the four possible keys; apple, banana, broccoli or none.
- The software needs to notify the user if an image has been labeled *none*
- These labeled keys need to be fed into the search algorithm
- The YouTube-8M database will need to be sorted into three lists, one for each key (except none).
- The given key will need to be coupled to one of the lists
- The selected list will print a sorted list of YouTube URLs.

In this project the following issues should be avoided:

- Using programming languages that not all group members are familiar with
- Using multiple programming languages that cannot be merged to create one integrated project
- Adding extra features that are not useful for the main goal of the project
- Bad communication that will cause inefficient information motion or waiting
- When the planning turns out to be not applicable we should not forget about the planning but adjust the planning

3.2 Software Architecture Draft

The project is split into two parts, creating the neural network for image recognition and using the YouTube Database to get all video IDs containing a certain object or topic. Consequently, we will split the work on these two features among the group members. If the two parts have been created, they should be integrated to work as one single searching tool, but this is only possible at the end of the project when both parts are finished. Hence, the two tasks should be implemented independently and our software should have high modularity. Within each module, high cohesion and low coupling are important as well. Figure 3 shows a first blueprint for the general architecture of our visual YouTube searching tool and how its two sub-parts are integrated with each other.

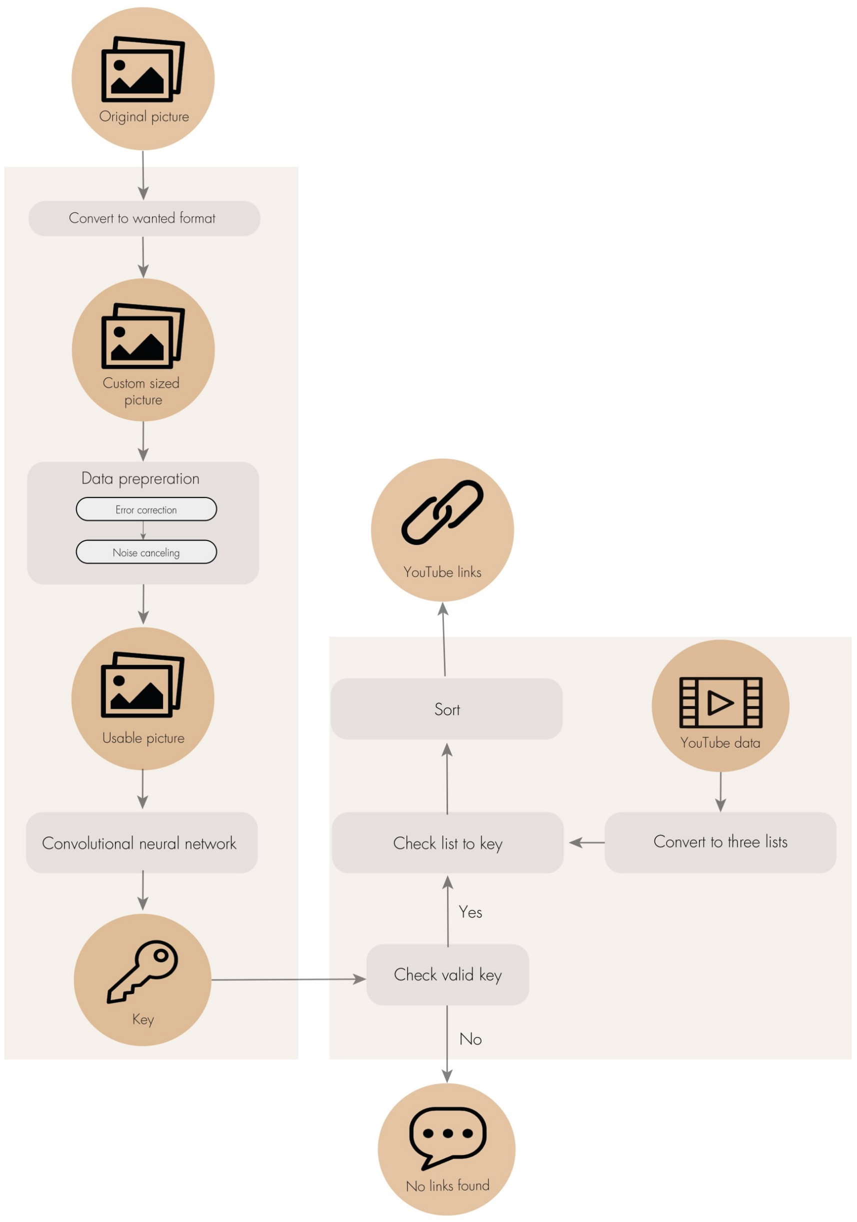


Figure 3: Initial software design diagram

3.3 Final Software Architecture

During the project some aspects of the initial software design plan were changed. Since using a pre-designed database set some limitations which ended up making the advantages not worth it. For example, the collection of YouTube 8M videos are of high quality, verified and labeled. However, these are external videos which need to be downloaded in order to be viewed and the provided labels were not specific enough for the purpose it needed to fulfill in the project.

Instead we decided to switch to a more direct approach where web scraping is used to directly request videos from the YouTube web platform and filter these on our own terms. Using the

YouTube platform directly came with its own disadvantages. For example, the videos collected were not always about the intended subjects. A fair amount of news, vlogs, TV shows and general entertainment topics needed to be filtered out. Also, the quality of each video needed to be evaluated. Lastly, the team came up with a design decision which would impacted the filters and thus the videos shown to the end user. This decision was to remove all videos that did not have their own thumbnail available to download as this was a necessity to have when displaying the video to the end user.

Next to that, different filter options were added for the user to be able to make the search more specific. Giving the option to filter on `relevancy`, `upload time`, `video length`, etc. gave the user more control over which videos they would see. Figure 4 visualizes the used software architecture for this this project.

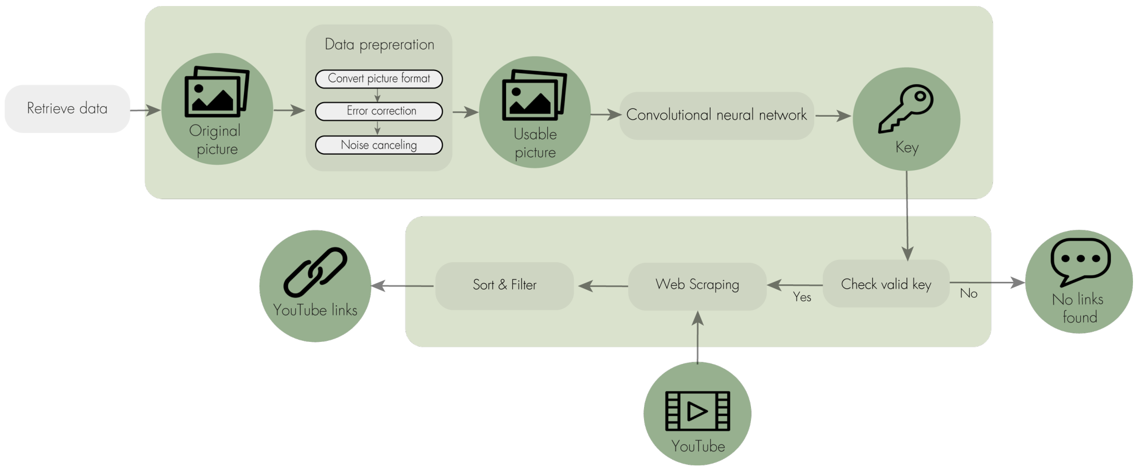


Figure 4: Final software design diagram

3.4 Implementation

3.4.1 Image Recognition

To implement and train a convolutional neural network, we first had to prepare the images of the COCO database to be fed into the network. For this purpose, we downloaded the training and validation images and annotations from the COCO website (`http://cocodataset.org/#download`). The training images, validation images and all annotations should each be saved in a separate directory. These directories should be stored in three strings in a script called `paths.py`. This is because each user of our network might save the data in different directories and these need to be known by our network for training.

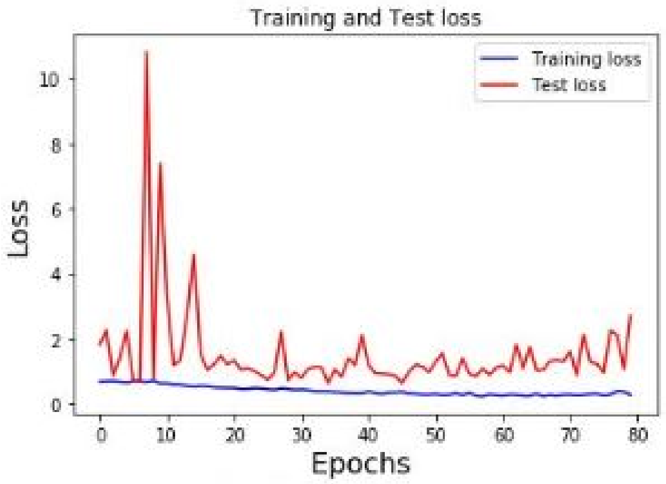
As the COCO database comes with a useful Python API for parsing the annotations, which are stored in *.json* files, we also downloaded the corresponding python file `coco.py` (`https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/coco.py`). The functions contained in this file helped us with reading the provided *.json* files efficiently and retrieve the file names and object class of each image in the training and validation samples. Hence, we created a file `data.py` that calls functions of `coco.py` and selects images of the desired object categories. Moreover, some pre-processing, such as only choosing images where the object is large enough was done in this file.

Next, we started to build the actual network. The architecture of our CNN is set up in the `network.py` script, which also contains functions for training and validating the network, plotting the loss and accuracy of the model and predicting the class of a given image. The main entry point, however, can be found in the `main.py` script, which connects the data and network scripts. The main can be run either in training or in prediction mode.

If the training mode is chose, the CNN will be trained all over again, which may take some time depending on the number of layers in the CNN and the amount of training images. Moreover, it is also possible to select a new threshold for the area that the object should fill in the training images and rewrite the resulting image names and labels to disk as *.txt* file. This can be done if the option to write is set to true and has a substantial influence on how well the network can be trained. If the chosen area is too large, only few images will fulfil this condition and too little training images lead to poor performance. However, if the chosen area is too small, the object of interest might be too small in some of the training images, leading to bad performance again. A value of 0.15 was found to be a good trade off between keeping a large amount of training images and a large enough object in each image.

After training a certain CNN architecture, the resulting weights and bias terms of the model are saved to disk in the *Models* folder of our repository. This is done to avoid long waiting times when making predictions. Hence, in prediction mode we simply load the pre-trained version of our CNN from the *Models* folder and predict the class of an input image using the stored model. This leads to fast predictions, but the quality of predictions depends on the quality of the model that was stored.

For The actual implementation we first experimented with relatively complex designs. Over five layers and use of multiple dropout and maxpool layers. This is inspired by the trend to use a lot of layers in current research in deep learning. This however did not give us good results. Shown in figure 5 are the best results we could get using a rather complex CNN. The accuracy is going up but it is instable. We could stop training when we notice that our accuracy is good on the specific training set. But since its so instable, it does not give a fair indication how it might perform on a different test set. Also a more complex model might try to learn details of the training pictures, when our goal is actually straightforward: classify banana's and broccolis.



```
:Figure size 432x288 with 0 Axes>
```

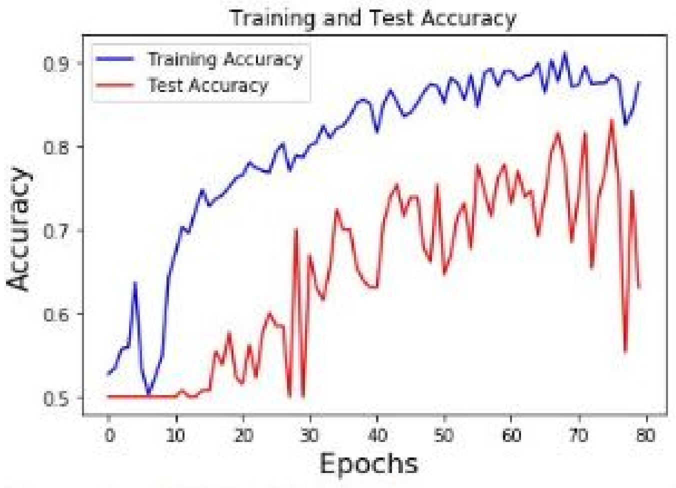
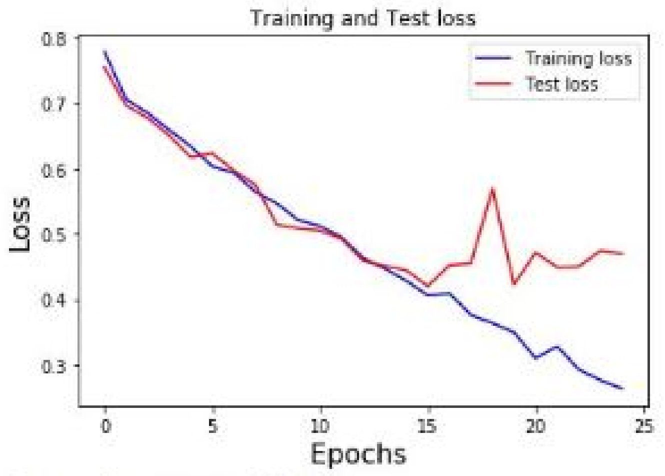


Figure 5: Best results using a complex CNN

After coming to this conclusion we tried to make our design simpler. finally we ended up using only 4 convolutional layers in total. The performance of this is illustrated in figure 6. As shown the accuracy is a lot more stable. Analysing multiple trained networks using this configuration we concluded a loss of about 0.40 is about the best we can do. That's why when we train our network we stop training when the loss is below 0.40 for the first time. This way we avoid overtraining the network on the training set.



```
<Figure size 432x288 with 0 Axes>
```

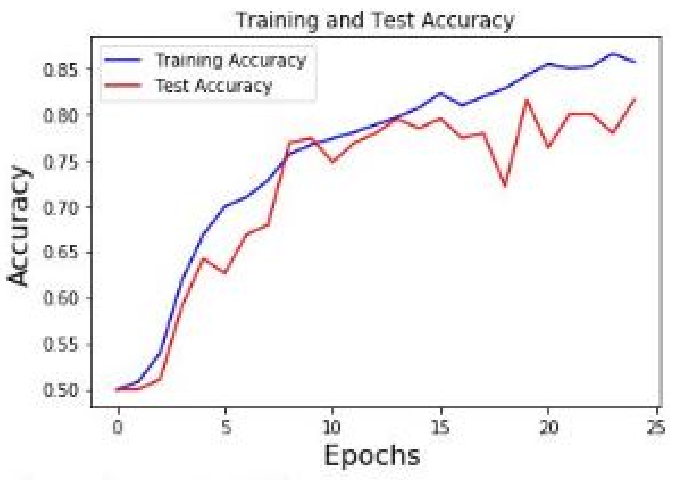


Figure 6: Best results using a simpler CNN

3.4.2 YouTube Data Retrieval

In order to search for YouTube videos given a certain search terms, a combination of two scripts are implemented. The first script is `VerifyRequest.py`, which checks whether there is a valid response at the given URL request. This script also limits the response wait time, so if the package is slow to return (over 2 seconds) the process is discarded and a new request is send. Furthermore, once the response package is received but has the wrong status code the function handles the exception by returning a `None`. The second script (`ModuleYT.py`) is a host of multiple functions that are build for a single purpose: requesting, receiving and filtering video hyper-links specifically for the YouTube web page. As the name suggest it is a module for YouTube. This means it is really easy take the interface of this module (the functional return values) and change it to fit any other web page and the rest of the system would still work. This is possible because the `ModuleYT.py` script uses one higher order function named *search_and_store* which only takes a search term and returns a list of URLs. Both the input and return variables of this function is web page independent.

What the `ModuleYT.py` does in depth will be explained in this section. When calling the *search_and_store* function we provide it with a search term which is the ingredient we want to search recipes for. The YouTube video request system works with a specific format for its search terms. So the input search term is first processed and formatted. Imagine the function was called with the term `"banana"`, the first thing to do is to clarify we are only intrested in recipe videos. Thus the term becomes `"banana + recipe"`. In case we want to specify more filters we can do that easily by adding `", <filter term>"`. Next, some characters need to be transformed for the URL to be valid:

```
"+" =>"%2B"
"," =>"%2C"
" " =>"+"
```

,

resulting in, for example, the following: `"banana+%2Brecipe%2C+short"`. To make the URL complete we add this term to the YouTube search query:

```
"https://www.youtube.com/results?search_query=banana+%2Brecipe%2C+short"
```

Once the search term format is correct it is time to send the URL request. For this the `VerifyRequest.py` is used to make sure the response is valid. If the verification returns a `None` type the request was invalid and the function will return an empty list with no video URLs. However, if the response is valid the response will be in the form of a web page. Here, *BeautifullSoup4* is used to retrieve all the hyper-links from the web page. This list of hyper-links can contain all kinds of links and not necessarily watchable YouTube videos. Because of this the list goes through some filters. The `Watch Only Filter` matches all hyper-links against a regular expression pattern:

```
"^(https?\:\/\/)?((www\.)?youtube\.com|youtu\.?be)\/watch.+$"
```

To look for hyper-links that contain the `"watch.+"` flag. The next filter is a bit more user focused. For the project it is important to give visual feedback to the end user. Thus the system requires a thumbnail filter. The `Thumbnail Only Filter` tries to retrieve the thumbnail for each video using its `ID` and the following URL format:

```
"https://img.youtube.com/vi/<ID>/maxresdefault.jpg"
```

Now the function is guaranteed to return YouTube links that are watchable and have an existing thumbnail. This is all done in real-time so no database has to be downloaded locally.

3.5 Testing Approach

The plan is to make the design really modular, i.e. split the design into parts and define which output belongs to a certain input. This way we can test the different parts of the system. When the code has been refactored, it is still possible to use those tests and thereby check if all the computations in the system are executed correctly.

Unit tests are a great way to isolate small parts of the system and test them more in depth. Our approach was to target code that was susceptible to breakage or failure. Thus the focus lays on web scraping rather than the convolutional neural network because the CNN is mostly based on the well tested tensorflow module. The web scraping however, is build up from scratch and thus more susceptible to flaws. The web scraping consists of three important operations: verifying requests send out to the web, processing YouTube's responds data with the goal of retrieve valid video hyper-links, processing each video id and retrieving its thumbnail image. There are 14 unit tests in total spread over these three subjects and test both 'happy' paths as 'bad' paths. In this case 'happy' means the reaction of a specific function on intended input and 'bad' means the opposite, the reaction on unintended input. This means these functions will also handle case exceptions and unintended inputs accordingly. These tests consists mostly of checking the response status codes and checking the types of input and output variables.

The final tests will be to provide images of objects that we took ourselves and check if the output is correct, so this will be entirely manual testing.