Table of Contents

1. Introduction

Video blogs, or as most people would call it, vlogs are content that would be found in a blog, which is traditionally text and images in video form. Generally, these videos are made by content creators to document or showcase activities in their daily lives of some special occasion. It is a huge contributor to traffic on video streaming sites like Youtube as they are the third most viewed type of video (Brown 2018). The scope of our project would be on a subset of vlogs called travel vlogs, where content creators upload videos about their travel experiences. These types of videos are apparently a huge influence towards people's decisions when travelling as 64% of people watch them when planning for a trip (Henderson 2018). Especially in recent years, Youtube statistics show that views for these types of videos increased by 41% between 2017 and 2018 (Travel video view statistics - Think with Google, 2018).

Most travel vlogs are long in length, most vlogs are about 20 minutes long. Therefore, as a viewer, most of the time we are only interested in a few specific landmarks or would like to skip over some sections of the video containing certain landmarks. A method that enables the viewer to do this is to add the landmark names along with the corresponding timestamps of which the landmarks appear in the video. This allows the user to directly seek to the position of the video where the landmark appears. However, as a Youtuber, as there is no automated process to do this, this has to be done manually. The Youtuber would have to watch the video and note down the timestamps individually which is an extremely time consuming video and is infeasible for a Youtuber who uploads videos frequently.

Those problems were the main motivation towards this project. Our aim and goal to achieve in this project is to solve the problems at both ends, the viewer and the Youtuber. For the Youtuber, we aimed to create a solution that will allow them to automatically generate accurate descriptions of the landmarks and timestamps in a reasonable amount of time. For the viewer, we aimed to allow them to detect landmarks from Youtube videos that are already online.

While searching on research papers that have done landmark detection, we noticed that all of the papers were doing landmark detection on images. We thought that it would be a breakthrough to implement a mobile application that can detect landmarks in videos instead. If successful, it would be a new contribution towards the field of landmark detection

on media. In essence, a video is a series of continuous images, thus, the fundamentals to traditional landmark detection approaches can still be used. However, we will be required to preprocess the video before passing it into the model as input, which would be the major part of our contribution.

In semester 2, we successfully created an Android app that achieves the main objective. The Android app allows the user to input a Youtube link and when the processing in completed, a list of clickable landmarks would be presented, allowing the user to click to directly seek to that timestamp in the video. The app also allows Youtubers to upload a local video file, which will be processed and a list of landmarks will be displayed as a long string of text. From there, the Youtuber can edit the description before uploading the video to Youtube.

2. Background

2.1. Introduction to literature review

For this literature review, due to lack of papers specific to our work, we would be looking at papers that are similar. Mainly, most papers that are published on the topic can identify landmarks in images, which is different from our goal that is to be able to identify landmarks in videos. However, in the low level breakdown of how our algorithm would work, we are fundamentally still attempting to detect landmarks in images. We would also be looking into papers related to other core functions in our project such as image augmentation, feature extraction and filtering frames.

The aim of this literature review is to be able to have a better understanding of existing works and approaches to this similar problem. By critically evaluating and analysing past works, we also aim to identify some shortcomings and hopefully be able to find a method to overcome it in our project to improve the overall performance.

2.2. Content

2.2.1. Landmark Detection Models

We have noticed that there are 3 main models that we can use as base models to use transfer learning in order to enable them to detect landmarks, namely the VGG-16 network, ResNet and Xception. We found that there has been contradictory research in the past showing that depending on the dataset used each of the networks perform differently.
However A. Nygaard in (Adil Nygaard, 2018) tests the three networks on the google landmark dataset and observers that VGG-16 performs the best achieving a test accuracy of 99% whereas its ResNet50 and Xception obtain 98% and 90% respectively. As we too are using the same dataset we shall use the VGG-16 model as, even though it performs worse on general image classification has been shown to perform superior in landmark detection by F. Chollet in (Chollet, 2017) and He et al mentions in (Kaiming He, 2016).

2.2.2. Blur Detection on Images

<u>Laplacian operator and OpenCV</u>

As our project requires us to have lesser images being passed to the test model, we need to have an efficient method to filter the extracted frames in terms of the amount of blur. This approach has been researched by R Bansal(2016), G Raj (2016) and T Choudhury (2016), where they are using Laplacian operator and OpenCV library. The Laplacian operator is being implemented from the OpenCV to perform calculate the amount of blur in the images.

The variance of the Laplacian kernel is calculated to determine the amount of edge-like features in the images. We will assume that a blur image will have very little edges in the image, thus if the variance falls below a pre-defined threshold, the image is considered as blurry. The method is simple to implement and scalable as the threshold value can be adjusted according to needs.

2.2.3. Efficient Frame Extraction

PySceneDetect

Instead of filtering the images based on a fixed intervals, we thought of another way which we can get the frames whenever a change of scene is detected. Scene Detection can be explained as splitting a video into sub-clips based on scene transitions. Some of the travel vlogs might contain scene transitions which are editing effects that are usually done with hard cuts, fades, and dissolves. There is an existing library in Python which automatically detects scene changes in videos. One of the detection types, which is content-based scene detection, is exactly what we require to do to our videos. According to the research paper that is done by DECKER, C. (2016), the content-based option checks for the differentiation between pairs of frames, which the value is being calculated by using heu, saturation and lightness (HSV) color space difference. The computed value is used to determine whether the difference exceeds a predefined threshold value.

2.3. Summary

In conclusion, having analysed several approaches for landmark detection in images, we have decided to use the modern approaches, specifically the VGG-16 model with transfer learning as we found that it consistently obtains the highest accuracy in such tasks. However, we will try to combine techniques in different approaches to attempt to improve the accuracy and performance further. In order to filter blur images, as it is not the main focus of our project, we will be using the Laplacian operator and OpenCV mainly due to its simplicity. Similar images would be detected using perceptual hash and usage of image key points due to its performance.

3. Methodology

3.1. Application

3.1.1. User Interface Design

The final application has 3 different activities, each representing a different part of the application. Google's Material Design guidelines were incorporated into the application to improve the aesthetic as well as the overall look and feel. Elements such as the rounded corners, the Roboto font style, material design text boxes and more. The constraint layout was chosen for all 3 activities mainly due to its simplicity in creating layouts in flat view hierarchy. Besides, guidelines were used widely in all views as well to ensure that the interface can be displayed on other devices with varying screen sizes and resolutions without issue.

3.1.2. Opening A Video File

In the upload to Youtube interface, a button is present in the top left corner of which when clicked, will allow the user to choose a file. To accomplish this, the button starts a new built in intent that starts an activity to allow the user to pick a video file from local storage. When the video is picked, the URI, which is the unique resource identifier that identifies the video file (which is returned by the intent) will be converted into an Android native file path (example: "/storage/emulated/0/Download/"). This conversion is done by using the query function in the content resolver, which returns a cursor that points to the path of the file. The path can then be used to play the video in the Video View as well as upload it to the server.

3.1.3. Viewing the Video in the App

For a video that resides in the local storage, to play it, the URI of the video file is passed into the setVideoURI function in the Video View of the interface. The media controls (play and pause) are enabled by creating an instance of MediaController and using the setMediaController function to set the MediaController to the Video View.

For a video on Youtube, the YouTube Android Player API was used to accomplish this. Firstly, the app is registered on the Google Developer Console to allow use to get an API key which was necessary for the API to function. Another prerequisite was that the activity that encompases the Youtube Player must extend the YoutubeBaseActivity class. From here, using the initialize function and passing in the API key and an instance of YouTubePlayer.OnInitializedListener() (which allowed the Youtube player to execute certain

commands once the initialisation is successful). When the “confirm” button is clicked, signaling that the user has entered a link, the id of the video, which is the random characters after the “=” symbol of the URL is obtained from the edit text by using a function. It is then sliced and passed into the cueVideo function of the YouTubePlayer, starting the video.

3.1.4. Downloading a Video From Youtube

In order to download a video straight from Youtube, a specific download link (different from the link of the video) must be extracted from the page. To achieve this, we used a library called YouTubeExtractor. This function takes the link of a youtube video and parses the webpage to return a download link. To specify the resolution, the library allows us to specify the itag value, which is a Youtube video stream format code that is unique to each resolution and file type, detailed in this link:
https://gist.github.com/sidneys/7095afe4da4ae58694d128b1034e01e2
As only the video is required to be processed, the function checks available itags and picks the itag with the lowest resolution without audio. A low resolution video is used to reduce the time taken to upload the video to Youtube.

Next, using the link, Android’s built in Download Manager is used to download the video. Using the Download Manager avoids the need to handle issues with the download such as loss of network connection, pausing and resuming the download, failed connection and more.

3.1.5. Uploading Video to Server

For this purpose, the OkHttpClient library is used. A client is first initialised. Then, the Content Resolver is once again used to access the actual video file, of which its location resides in the Downloads folder if detecting from an existing Youtube Video and in the path specified by the user for local videos). An HTTP request body is created which contains:

(1) The size / length of the file
(2) The type of file, which in this case is fixed to mp4
(3) The actual file

The request body is split into multiple parts for transmission using the MultipartBody.Builder() function. Finally, an HTTP request is created, of which the previously created request body is attached and sent to the server. When a response is received (the server responds with the list of landmarks and timestamps when the processing is completed), the landmarks and

timestamps are then parsed to either be displayed as text in the upload to Youtube section or a list of landmarks in the existing Youtube video section.

3.1.6. Clickable Landmarks in the List View

To implement the functionality where each item in the list is clickable, a list view with array adapter is used. When the response (which is the landmarks and timestamps sent from the server) is received, it is parsed into 2 Array Lists, which is a suitable data type as it's size is dynamic:

(1) landmarks_list - an Array List of strings where each index contains the timestamp (in hours, minutes and seconds) and the name of the landmark as a single string.

(2) timestamps - an Array List of integers containing the timestamps in milliseconds of where each of the landmarks appear in the video.

The indexes in both Array Lists match, meaning an index in the landmarks_list would correspond to the same landmark as the same index in the timestamps list. Using the setOnItemClickListener method in the ListView, each time an item is clicked in the list view, the position of the item clicked would be used as the index to obtain the timestamp in milliseconds of where the landmark exists in the video. From there, the Youtube Player's seekToMillis function is called and the time is passed in, making the Youtube video seek directly to that position.

3.2. Server

The server is implemented using a Python Framework called Flask. This was chosen as most of the codes written for this project where in the Python language, making the integration process slightly easier. In the code for the server, the function handle_request will be called whenever there is an incoming request to the server. In this function, flask.request.files is used to get the incoming file (which is the video that is being uploaded). Werkzeug.utils.secure_filename is used to obtain the correct file name for the video. The video file is then saved to the current directory. Finally, the main() function, which contains all the codes to process the video (explained the sections below) is called and the result of the function is returned as the response.

3.3. Video and Image Preprocessing

3.3.1. Frame Extraction

Initially, our proposed plan was to use the traditional method to extract frames based on a fixed interval, which can be set by us according to the videos. The mentioned method can be done in Python by importing the OpenCV libraries, where the specific function that we are using is cv2.VideoCapture(). The path to the video file needs to be specified as the parameters of the VideoCapture function to create a video object. The video object is then being used to read the videos and write the frames being read from the videos into images by using cv2.imwrite().

However, throughout the semester, we have found a better method to extract frames from the videos, which is another Python library called PySceneDetect. It is also a command-line application which is able to split the video into separate clips automatically. The way PySceneDetect executes is to detect whenever there is any scene changes, then extract the frames based on the changed scenes. There are two modes available in PySceneDetect, which are Content Detection and Threshold Detection.

The general command line version of PySceneDetect to extract frames from video is:
scenedetect -i <video_filename> detect-content list-scenes save-images

We have chosen Content Detection to fit this function into our project, where one of the sub-options under save-images called --num-images N is being used to adjust the number of frames being generated at each detected scene. As we are running our model on server, we have to reduce the number of frames to reduce the workload on the server. We decided to set the number of frames extracted at each scene to be 1. Thus, the final version of the command line that we have used is shown below:

scenedetect -i <video_filename> detect-content list-scenes save-images -n 1

However due to server issues, we have modified our code to use the Python interface of PySceneDetect instead of the command line interface. However, the functionalities and implementations are similar as mentioned above.

3.3.2. Blur Detection

Blur Detection is implemented to further cut down the repeated and blurred images which are unnecessary to be passed into our training model. It can be implemented by using OpenCV library in Python, mainly by processing the image in the matrix form.

Similar to other image processing methods, an image is preferably to be converted into a grayscale image which only consists of single channel if an image. The grayscale image is then being convolved into the 3 x 3 matrix, known as Laplacian kernel. The second derivatives of the image is then being calculated by using Laplacian operator. Laplacian is able to emphasize the region which have high intensity changes, which in a way managed to indicate the amount of edges contained in the image.

The variance of the Laplacian kernel is being calculated to find out the area of spread for the responses. If the variance falls below a predefined threshold, then the image is considered blurry because we know that a blurry image normally has less edges compared to a high quality image.

We have to decide the threshold by ourselves depending on the images generated. In our final product, we have used a threshold of 300, which works fine and able to detect and filter out really blurred images. It does, however, eliminates those images taken with Depth of Field technique because those images has relatively lesser edges. We did not regard this as an issue because it is better to filter out the images without landmark being focussed too. The images which are being detected as blurry will be deleted after running the script.

3.4. Model

3.4.1. Data sets used

The google landmark dataset with over 2 million images and 30 thousand unique landmarks was to be used, however, since the landmarks were only given an ID instead of being represented by their respective names, it required augmentation. There were 2 methods in which the names of landmarks could be added:

1. Write a script to crawl the web/google images- However it was noticed that this would cause noise leading to inaccuracy in our dataset and thus create potentially large inaccuracies in our model.
2. Manually identify IDs with their corresponding names – This approach was used, however, due to the amount of time required for the manual work, it was infeasible to augment the entire dataset. Thus, we resorted to selecting the landmarks that contained the greatest number of images in the dataset and separated them out. Using the top 50 landmarks gave us a total dataset of over 275,000 images divided into 215,000 train images, 55,000 validation images and 2,700 test images.

3.4.2. Image Augmentation

Each image that was downloaded was used in a 96x96 dimension in order to reduce its resolution thus making the training and more importantly the prediction using the model much faster. When training the data images are augmented at random to create more variance and in order to achieve this a rotation range of 90 degrees, a width range of 0.2, a height range of 0.2 and a zoom range of 0.2 is used. This will allow for the images to be shifted and allow for a better model.

3.5. Model Training

3.5.1. Base Model

The base model used was created using the VGG16 architecture trained on the ImageNet dataset. The train data as well as the validation data is run through this base model using a batch_size of 1000 for 5 epochs in order to convert the images into vectors using the weights from the ImageNet on VGG16.

3.5.2. Transfer Learning

Three layers were added on top of the VGG16 model, which comprised of two dense layers that used relu activation and a final dense layer with softmax activation. The rmsprop optimizer was used as it provided the best results. The 3 layers were initialized with weights by fitting the model with the train and validation data with a batch size of 1000 and run for 10 epochs.

Following this the complete model was generated by combining the base model and the top 3 layers. The first 16 layers are set to be made untrainable thereby making the weights fixed in order to ensure the weights from ImageNet are not lost. The model is compiled using the Adam optimizer with a learning rate of 0.0001 and decay of 0.0 as these were realized to be the optimal values following a few tests on smaller datasets. This model is trained using a batch size of 500 and run for 50 epochs with each epoch having 340 train steps and 150 validation steps.

3.5. Predictions

Images are retrieved one by one and are run through the model provided that they are not blurred. By removing the blurred images, only the images with distinguishable features are passed to the model, thereby increasing the accuracy of the model. Once the scores for each classification are generated from the model. An argmax function is used to get the maximum score along with its corresponding class. The score is then checked to see if it is above 0.7. This is done in order to ensure that only the images that the model is most confident about are detected as landmarks. Once detected, the data is sent back to the client.

4. Project Management

4.1. How was the project conceived and managed

During the first meeting in semester 1, the main agenda was to discuss the topic to be done for the Final Year Project. Prior to this, research on different fields such as steganography, security and machine learning, were done separately by each of the team members. The topics suggested by each member were as shown below:

1. Steganography to survive through the communication between different Social Networking Service by Nicholas
2. Copy-move Forgery Detection on Images for Forensics by Si Qi
3. Landmark Detection on Images by Nikin

After some discussion, we decided to choose Landmark Detection as it is an interesting topic that the rest of the teams have never tried before and we are interested in trying. In addition, each of us have experience in the components of the project. Nikin took a Deep Learning unit previously, which means he has the expertise to complete the major part of this project, which is the landmark detection part which involves machine learning. On the other hand, Nicholas and Si Qi have taken Mobile Application and some Java related coursework, thus we are quite confident that we could build a mobile application aside with the landmark detection algorithm.

As we proceed to discuss about details to work with Landmark Detection in Images, Nicholas has suggested that we should do videos instead of images. It is not just because it will be more challenging, but it comes with more meaningful objectives, which are:

1. To enable Youtubers to upload their travel vlogs with automatically generated annotations of landmarks in a much shorter time
2. Provide an easier way for the Youtube audience to identify the landmarks in existing Youtube travel vlogs

After further research, we did not manage to find any research papers regarding landmark detection on videos, only landmark detection on videos existed. However, we were still keen on going with this as we felt like it would be a challenge for us personally as well as a contribution to this field.

4.2. Project Management Approach

The lifecycle model used in the actual development of the project was agile as proposed in the proposal. The main benefit we found by using this approach was the flexibility towards changes mid-way of the project (Gilley, 2015). During the project, there were several changes that were made due to the challenges faced. The biggest issue was regarding the lack of landmark names in the Google Landmarks Dataset. At that point of time, we have already completed the analysis and design and are currently working on the code. If the waterfall approach was used, we would need to restart the previous two stages and would therefore be a major setback towards the progress of our project. By using the agile approach, we were able to make a few adjustments towards the goals and design of our product without much setbacks in the schedule.

Another advantage that was greatly felt in using the agile approach was the ability to access the risk and value in small increments of the project (Kerzer, 2017). The agile approach forces us to check the feasibility and risk throughout the duration of the entire project to determine if this our current approach towards the problem is working or we should pivot to try an alternative. This is seen when we were attempting to integrate the model, frame extraction and filtering of frames into our app. We first tried to run the model directly on device. However, after trying for a period of time, after assessing the risks and the possible gains, we felt that this approach was not worth pursuing anymore and pivoted back to the server approach. If such an incident occurred in a waterfall methodology of project management, the design and analysis plans for the project would have to be scrapped and revisited, wasting a lot of unnecessary time.

Lastly, due to our lack of knowledge in some of the technologies that were required to accomplish this project, the agile model's leniency on specific project requirements helped us greatly. Often times during the project while implementing a certain part, we realised that we needed certain libraries or APIs that were not initially expected in the proposal. With this flexibility, we were able to go back and make small changes to the proposal without much changes in schedule.

However, one disadvantage that comes with using the agile approach is the lack of proper planning of resources according to the article by Planview LeanKit. This caused some roadblocks in semester 2 as we did not anticipate the need for certain resources. For example, in the proposal, we proposed the use of Amazon Web Services server for training the model as well as hosting it. However, we soon discovered the cost incurred to use the AWS server to train the model is too high to be feasible. We were then forced to use the lab

machine for this purpose. Had we calculated and estimated the cost of renting the server beforehand, this problem would not have occurred.

4.3. Project resources, execution and planning

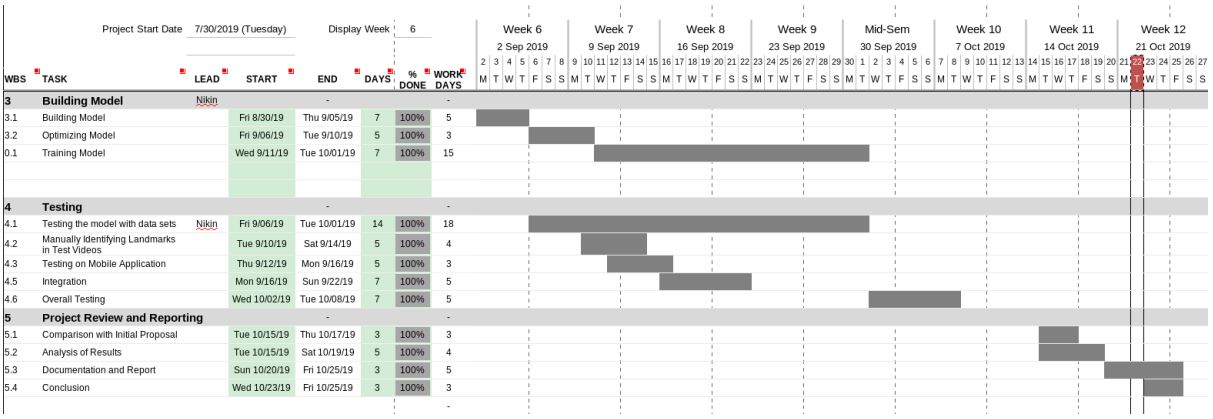
4.3.1. Project Schedule

[Team 8 FYP: Landmark Detection in Videos] Project Schedule

[Monash University Malaysia]

Project Start Date: 7/30/2019 (Tuesday) — Display Week: 1

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
1	**Pre-processing**			-			-
1.1	Extracting Video Frames	Si Qi	Mon 8/05/19	Sun 8/11/19	7	100%	6
1.2	Blur Detection		Wed 8/07/19	Sun 8/11/19	5	100%	3
1.3	Data Set Labeling	Nic	Mon 8/12/19	Thu 8/15/19	4	100%	4
2	**Mobile Application Building**			-			-
2.1	Basic User Interface	Nic	Thu 8/15/19	Sun 8/18/19	4	100%	2
2.2	Youtube APIs Related		Mon 8/19/19	Fri 8/23/19	5	100%	5
2.3	Video and Image Related	Si Qi	Mon 8/19/19	Fri 8/23/19	5	100%	5
2.4	OpenCV		Mon 8/19/19	Sun 8/25/19	7	100%	5
3	**Building Model**	Nikin		-			-
3.1	Building Model		Fri 8/30/19	Thu 9/05/19	7	100%	5
3.2	Optimizing Model		Fri 9/06/19	Tue 9/10/19	5	100%	3
0.1	Training Model		Wed 9/11/19	Tue 10/01/19	7	100%	15



Project Start Date: 7/30/2019 (Tuesday) — Display Week: 6

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
3	**Building Model**	Nikin		-			-
3.1	Building Model		Fri 8/30/19	Thu 9/05/19	7	100%	5
3.2	Optimizing Model		Fri 9/06/19	Tue 9/10/19	5	100%	3
0.1	Training Model		Wed 9/11/19	Tue 10/01/19	7	100%	15
4	**Testing**			-			-
4.1	Testing the model with data sets	Nikin	Fri 9/06/19	Tue 10/01/19	14	100%	18
4.2	Manually Identifying Landmarks in Test Videos		Tue 9/10/19	Sat 9/14/19	5	100%	4
4.3	Testing on Mobile Application		Thu 9/12/19	Mon 9/16/19	5	100%	3
4.5	Integration		Mon 9/16/19	Sun 9/22/19	7	100%	5
4.6	Overall Testing		Wed 10/02/19	Tue 10/08/19	7	100%	5
5	**Project Review and Reporting**			-			-
5.1	Comparison with Initial Proposal		Tue 10/15/19	Thu 10/17/19	3	100%	3
5.2	Analysis of Results		Tue 10/15/19	Sat 10/19/19	5	100%	4
5.3	Documentation and Report		Sun 10/20/19	Fri 10/25/19	3	100%	5
5.4	Conclusion		Wed 10/23/19	Fri 10/25/19	3	100%	3

4.3.2. Project Resources

Tasks	Resources Type	Resources Details
Training Data Sets	1.Equipment	A workstation computer was provided for training the model Model: Dell Precision T5820 workstation Operating System: Ubuntu Linux CPU: Intel® Xeon(R) W-2145 CPU @ 3.70GHz RAM: 62.6GiB
	2.Technologies	i. Programming Languages **Python** because it is a high level language for machine learning and easier to use ii. Platform Pycharm Community Edition iii. Libraries **Keras** allows fast experimentation of neural networks and modularity **TensorFlow** will be used on building models by using Keras with eager execution to achieve immediate model interaction. **HTTP** which is the Python library to implement the GET and POST method to obtain information from the server side **Google API Client Libraries** offer a simple and flexible access to Google APIs v. Server Side Language **Python** is used to establish a connection between the server and users

	3. Materials	**Subsets of Google Landmarks Data Sets** which contains 275,000 images, with 50 unique landmarks
	4. Time Taken	Building the initial model: one week Training the data sets: one week Training with different settings: two weeks Total: 4 weeks
Matching and Evaluating Data Sets	1. Technologies	<u>Libraries</u> **PySceneDetect** is used to extract frames based on the number of scenes detected to reduce the number of repeated frames. **OpenCV** to optimize the code and filter out blurry frames which will then be sent to be trained
	2. Time taken	Filtering frames: one week Optimization of code: one week concurrently Data Set Labeling: 4 days Total: < 3 weeks
Building the mobile application	1. Technologies	i. <u>Programming Languages</u> **Java** will be used for building mobile application because it is the native language for an mobile application ii. <u>Platform</u> Android Studio
	2. Time taken	Basic functions: 5 days OpenCV: one week

		Improve user interface: one week Total: ~ 3 weeks
Testing the application	1. Technologies	Uses **Youtube API** to add a function into the mobile application to be able to upload the videos directly from the mobile application to the Youtube platform **Python Web Framework, Flask,** will be used for connection purpose between server and users
	2. Time taken	Manually testing the results with selected videos: one week Testing the mobile application: one week Total: 2 weeks
Project Review and Reporting	1. Labor	All team members should keep track of the progress throughout the entire project. A record of important decisions and steps taken should be maintained over the period of the developing process.
	3. Time required	Documentation and analysis: one week Comparison with proposed results and Conclusion: one week Total: 2 weeks

4.4. Risk management

For risk management techniques, our team initially had a team meeting to brainstorm potential risks that might arise from the project. Besides, SWOT analysis, which is short for “Strengths”, “Weaknesses”, “Opportunities” and “Threats” will be applied on the project. By determining strengths and weaknesses, we identified the risks in the context of opportunities and threats.

We maintained a risk register which was constantly updated as we faced risks and resolved them. The updated risk register following the completion of the project is shown as Appendix A.

4.5 Limitations encountered during project management

4.5.1 Time Management

During the course of the project, we realised that we were required to be constantly rescheduling our work due to roadblocks in the development stage and therefore lead to slight delays and inefficiencies. Even though it must be noted that some scheduling changes are inevitable, it would have been more efficient if we were able to continue with a schedule as planned without any changes throughout the course of the project.

4.5.2 Versioning

Mid-project, it was realized that there was a need for proper version control in the files that were used in collaboration between team members and thus a github implementation was used from that point on. However, the use of such a system from the start of the project would have been beneficial and would have resulted in a large time-saving when collaboratively writing code.

4.5.3 Communication

We realized that one of the main lapses in project management was the communication aspect. Due to this we faced complications towards the end of the project when trying to integrate the system as some of the dependency libraries used in certain sections of code were not supported for others. However we were able to overcome this with minimal effect on the final project and without extensive delays through the improvement in communication between team members.

5. Outcomes

5.1. Results achieved/product delivered

We have managed to obtain some achievements on our final product delivered throughout the semester, which they will be discussed in the sections below.

5.1.1. Accuracy of Model

One of our major achievements is that our model managed to detect landmarks in the test videos correctly at a high percentage. We have a validation accuracy of 90%. Our testing stage was being carried out on approximately 20,000 landmarks, where our model is being specifically trained to be detected the 50 landmarks in our selected list. For all the landmarks listed under the selected list of 50 landmarks, there is a 90% of accuracy on validation accuracy.

5.1.2. Filtering of Frames

5.1.2.1. Frame Extraction from Videos

In our proposed method, the video frames will be extracted based on fixed intervals. It was being set to be extracting one frame for every 25ms, which there will be 40 frames per second. By having a 48 seconds video, originally 1447 frames will be extracted by using the traditional method. However if PySceneDetect is used, there is only 17 scenes being detected. Number of frames to be extracted at each detected scene can be adjusted by changing the parameters. A minimum frame number of 17 could be achieved if we set the parameter to be extracting one frame per detected scene. This method has significantly reduced the number of repeated frames being sent over to the model to be trained.

5.1.2.2. Blur Detection on Extracted Frames

The initial objective of implementing blur detection is to cut down the number of frames. The model will not be able to detect any landmarks from blurred images, thus it will be more efficient if we could remove them at the first case. A threshold is required to be set, where any images that has a lower value than threshold will be removed. After a few testing and adjusting the parameters, we managed to get the optimum threshold value, where the images that passed the blur detection are mostly clear enough for the model to detect.

Overall, our final product managed to filter out most of the extra repeated scenes automatically without adding any adjustment for every run. The number of frames that are being sent over to the model are being reduced to more than 50% lesser compared to our previous method. This is considered a major achievement because we can use the extra slots to extract at two frames per scene to increase the accuracy to detect the landmarks.

5.1.3. Integration

We have different parts that is being implemented separately using different programming language and will be combined as a single product at the end. The landmark detecting model can only be run on Python by using Tensorflow and Keras libraries, where the mobile application is preferred to be implemented on Android Studio because the native language for an Android application is Java. The original plan is to run the model directly in the mobile application, however failed due to too much dependency on the library used in the model. We managed to integrate both of the results done in two different platforms together as a final product by hosting a server and running the model on the server.

5.2. How are initial requirements met

Requirement	Met (Yes/No)	Notes
Satisfactory model accuracy	Yes	Acheived 90% validation accuracy
Detect landmarks in videos in a reasonable amount of time	Yes	
Clickable list items to directly seek to that timestamp in the video	Yes	
Viewing of the video in the app while processing	Yes	Achieved using the Youtube API
User friendly mobile application	Yes	Final product is simple and easy to use
Able to upload videos from device	Yes	
Able to download videos from URL	Yes	
Scalable approach used	Yes	
Download and upload video files in a reasonable amount of time	No	Download and upload speed is unsatisfactory and dependant on the connection speed
Able to detect a wide range of landmarks	No	Restricted to 50 landmarks due to issue with dataset
Able to directly upload video to Youtube with descriptions	No	Unable to integrate Youtube API in the app
Compatible with many file types	No	Restricted to mp4 format

5.3. Justification of decisions made

5.3.1. Running the model on device vs on a server

Initially during the proposal phase, we proposed that we would be sending the video file to a server to be processed. However, after assessing the performance and time taken to process a frame in the video which turned out to require much less processing power and time than we expected, we attempted to run the model on the device itself. The advantages of going with this approach are:

(1) The model can be stored on device, allowing the application to run without the access to Internet.
(2) Uploading the video to the server is no longer required, therefore significantly reducing the runtime of the entire process and allowing the file size of the video to be much larger.
(3) A server is no longer required to be kept online to use the application, reducing the cost of operation of the app.

We tried using Chaquopy, which is a Python SDK for Android that allows a single Android app to simultaneously run Python, Java and Kotlin code. The first issue we faced was regarding the model. In order to run the model, many libraries are required, such as Keras, Tensorflow, numpy and more. Chaquopy lacked support for some of these libraries. We could find alternatives to these libraries, however, we found another issue, which is that the size of the resulting application was close to 1GB even when the file that stores our model was only around 60MB. A present day Android phone has and average of around 64GB of internal storage, making a 1GB app impractical.

Due to this, we revisited the server approach. To ease integration as all of the code for our model runs on Python, we decided to use a web framework that is built on Python called Flask. The advantages of using a server are:

(1) If the server is a machine with enough computing power, the time taken to process the video can be drastically decreased.
(2) A server with more computing power would allow us to tune certain parameters in our model such as the number of frames taken per scene in PySceneDetect and the blur threshold in the Blur Detection using OpenCV, improving the accuracy of our model.
(3) The processing of the video, if run on device, would cause a significant drain of battery as it is a CPU intensive workload. Running the model on the server would reduce the battery usage of the application.

Therefore, we decided to implement the server approach.

5.3.2 Learning rate and optimizer of model

It was previously proven through multiple research papers as well as our own tests that the adam optimizer was the best at producing results in improving the accuracy of the model and therefore it was used.

For the learning rate it is necessary to have a low rate in order to improve the accuracy of the model, however if the rate is set to be too low it may result in exponential rises in the time taken to train the model. Therefore considering the constraints, we found 0.0001 to be the optimal learning rate to find a balance between accuracy and the time taken.

5.3.3 Number of Epochs and Batch Sizes

The two main requirements from varying these two variables is to obtain a model that is accurate as well as one that does not take too long to train. Thus it is necessary to make sure that the two values need to be high enough to make the model accurate but not too high as to make it take too long to train. We needed to test various values and find the sweet spot at which the increase in accuracy did not justify further increase in values due to the time factor.

Following various tests it was found that a batch size of 500 and 50 epochs was the ideal.

5.4. Limitations of project outcomes

5.4.1. Lack of User Friendly Functions

Due to our server framework, which only allows for 1 response per request, we were unable to implement a working progress bar in the application. This negatively impacts the user experience as there is no indicator of the current progress of the processing as well as the estimated time remaining. The user will only be presented with the resulting landmarks and timestamps when all the processing has been completed.

5.4.2. Limited Number of Landmarks

Due to the issue we faced in regards to the Google Landmarks Dataset, which was that it did not contain the names of the landmarks, only the IDs to identify them, we were forced to manually match the IDs with their corresponding landmark names. This process was very time consuming and therefore we limited the scope of the landmarks to the 50 landmarks with the most images in the dataset. Therefore, although we were able to achieve a rather high accuracy when tested on images containing one of the 50 trained landmarks, if an image which contains a landmark that is outside the scope, one of two things might happen:

(1) The landmark will simply not be detected at all
(2) The landmarks will be falsely detected as one of the 50 trained landmarks that might look similar. For example, a castle in the image is detected as one of the castles in the 50 trained landmarks.

5.4.3. Network Connection Speed

As the entire video file has to be sent to the server for processing, the upload and download speed at both the server and the user's device is a major factor towards how long the user has to wait before he or she can get a result. Therefore, if the video file being uploaded by the user is high quality with a high resolution and frame rate, the file size might be huge which causes it to take a long time to upload, negatively impacting the user experience. To mitigate this, the video downloaded from Youtube will be downloaded at the lowest resolution possible and without the audio track. However, from our testing, a Youtube video with a longer duration will still take a substantial amount of time to download.

5.4.4. Lack of Youtube Integration

This was one of the features that we failed to implement. Initially, in the proposal, we proposed that the app have a functionality that allowed the user to directly upload the video

from the local storage, along with the detected landmarks and timestamps directly to Youtube. However, we failed to implement this due to issues with integrating the Youtube API in our application.

5.4.5. Compatibility

As of now, the only video file type that is accepted in our application is mp4 (MPEG-4). This is due to the way the uploading and downloading is done in both the server and application side. Furthermore, the current implementation of the detection algorithm is only tested to work on mp4s.

5.5. Improvements and possible future works

Find a dataset with tagged landmarks and make a more inclusive model.

By using a larger model that consists of a larger array of landmarks, it is possible for us to make the model more inclusive thus increasing the overall accuracy of landmarks detected by reducing the false positive rate when the model is fed with other landmarks. The current method used to train can be easily scaled up for this purpose provided that a larger data set is found as well as the required computational resources are available.

Incorporate Deep Local Feature(DeLF) Extraction to reduce false positives

False positives are currently generated when images without landmarks in them are sent into the model, even with a higher threshold for detection. Therefore, through the incorporation of DeLF extraction and using it as a final crosscheck mechanism to the models prediction, it is possible to reduce the number of false positives generated. However this may have computational time implications and therefore may require more powerful servers to run the prediction.

Increase compatibility in app platforms and video types

Currently the system is only implemented for the android platform, and thus reduces the user-reach and by expanding it to other systems such as IOS and windows more widespread user adoption could be expected. Moreover, through the integration to browser extensions such as chrome extensions, it is possible to improve the user experience as well by preventing the need for users to use a completely new app.

Furthermore, currently our system only accepts mp4 files and by increasing the amount of file formats, it is possible to improve the user experience.

Use of a database to reduce runtime

By using a database to store the outputs of a given video file, it will be possible for us to use the data from the database directly the next time the video is searched for,thus reducing the time taken to run the prediction model. This may be implemented as a future improvement to our system.

Further integration with youtube

Currently the system is not possible to directly upload the videos with descriptions into youtube, however with further development features such as but not limited to this may be incorporated into the application thus making it more user friendly.

6. Conclusion

In conclusion, our project manages to provide a proof of concept on the ability for current technical advances in machine learning to detect and identify landmarks in videos. The work, while being built upon a few research papers such as Nygaard in (Adil Nygaard, 2018) and F. Chollet in (Chollet, 2017) that propose solutions to landmark detection in images, have furthered this field as an initial dive into videos. The project provides a method which is highly scalable to allow for extension into a larger dataset as well as for public adoption given sufficient technical resources.

The project, while did have initial setbacks as well as certain goals unachieved, successfully achieves all the major objectives that were set out at the start.

7. References

Adil Nygaard, U. N. (2018). Final Project. Google Landmark Recognition Challenge. Stanford University.

Brown, C. (2018). Here Are the Top 10 Most Popular Types of Videos on YouTube. Retrieved from mag.octoly.: https://mag.octoly.com/here-are-the-top-10-most-popular-types-of-videos-on-youtube-4ea1e1a192ac

Decker, C. (2016). Scene By Scene Script Generation For Live Action Hollywood Movies. (Doctoral dissertation, Stetson University).

Gilley, C. (2015, 03 12). *The Pros and Cons of Agile Product Development.* http://community.uservoice.com/blog/the-pros-and-cons-ofagile-product-development/

Henderson, P. (2018). Video Marketing in Travel - 10 Key Insights. Retrieved from markgrowth: https://blog.markgrowth.com/video-marketing-in-travel-10-key-insights-a64fdcffa465

Kerzner, H. (2017). Agile/Scrum Project Management. Project Management Case Studies (5th Edition).

Travel video view statistics - Think with Google. (2018). Retrieved from thinkwithgoogle: https://www.thinkwithgoogle.com/data/travel-video-view-statistics/

Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. CVPR, 1251-1258.

Planview Leankit. (2019). *What Are The Disadvantages of Agile?* https://leankit.com/learn/agile/what-are-the-disadvantages-of-agile/

Raghav Bansal, G. R. (2018). Blur image detection using Laplacian operator and Open-CV. 5th International Conference (pp. 399-403). ICEEE.