IT Carlow –

BSc.

Software Development

UAV using Convolutional Neural Networks

Final Report

Student Name: Josh Hudziak

Student Email: C00231846@itcarlow.ie

Date: 30/04/2021

Supervisor: Dr. Oisin Cawley

Supervisor Email: oisin.cawley@itcarlow.ie

Table of Contents

Table of Figures

1 Abstract

The purpose of this project was to create an autonomous UAV, that could navigate through roads, paths and tracks. This technology is becoming more prevalent with the emerging UAV technologies around the world. Autonomous UAVs have use cases in search and rescue, medication delivery, military surveillance as well as surveillance in the construction sector. This autonomous navigation is possible thanks to cutting edge research in deep learning.

2 Introduction

This document's goal is to give an overall overview of the work that went into making this project, the final design of the project and problems that were encountered and how they were overcome, as well as, what was learned during the project.

The end goal of this project was to create a fully autonomously navigating UAV using a CNN (Convolutional Neural Network). There are many ways to train a network like this and many tools to do so, not to mention the hardware requirements to make this project functional, all these aspects will be reviewed in the project overview section.

A section of the document will be dedicated to reviewing the changes in implementation and design that occurred from the original research and functional document.

Another section will review problems that were encountered with the design, hardware and software during the implementation. The learning outcomes as well as technical outcomes of the project will also be reviewed here.

3 Project Overview

3.1 The Dataset

A major milestone to the implementation of this project was to gather a dataset that a model could train upon and then predict UAV footage live. This prediction would be what naviagets the UAV around its path, road or track.

This dataset was collected using a One Plus 7 phone camera and a Akaso 7LE camera. The first datasets that were gathered were taken from one point of view. The footage was first split into frames using a python script. These frames were then segregated into categories: left, right, centre. The frames being assigned to a category was based upon if that frame was taken during a turning event.

Problems occurred with this method which will be explained in the problem section of this document. Due to these issues a new method was implemented. The next method to gather data was to take separate footage for each category of data. One category would have footage where the camera was turned slightly to the left, This footage would be assigned to a single category and the method would be repeated for the centre and right categories as well. The angle of which the camera could be pointed in a car was limited by the windshield but while walking the camera was set to 30° to the left, 30° to the right or straight in the middle for the centre category. This method also required numerous trips of the one route as only one camera was available and could be mounted.

Figure 1,2,3: The camera setup used for tracks and paths, The direction the camera points in either direction being -30° from the centre. 1-centre, 2-left, 3-right.

Figure 4,5,6: The different results from the 30° angle position method. 1-centre, 2-left, 3- right.

Once the dataset had been gathered, preprocessing was implemented to reduce the data flow into understandable forms for the model. First the data was split into subsets called training, validation and test. This split was 80% training, 10% validation and 10% test. This provides a test for the model, when it has been trained upon the training set, the model is evaluated on the validation set and further evaluated upon the test data. As the data set grew much larger in the later stages there was no need for a validation set and the subsets were changed to 90% train and 10% test.

In the beginning data augmentation was used to create a more fruitful dataset to train on. This augmentation would rotate, blur and shift widths and heights of the images. After much testing it was found that this augmentation was hampering the models accuracy. Thereafter the data was just rescaled to smaller resolutions. The data flowing from these categories is packed into batches which the model then takes and learns upon, these batches are randomly shuffled to improve generalisation and to avoid overfitting.

3.2 The Model

Using an array of new technologies such as Keras and Tensorflow a classification prediction model was created. This model was trained on the dataset mentioned previously and thereafter tested on the test subset of that dataset.

3.2.1 Tools and Libraries Installation

These Tools are run in an anaconda environment and all coding is done using Python and various Python libraries in Jupyter Notebooks. In the anaconda shell install Jupyter Notebook: `conda install -y jupyter` To install Tensorflow in anaconda, create a virtual environment: `conda create --name tensorflow python=3.7` and activate this environment: `conda activate tensorflow` Next installing the tensorflow libraries use: `conda install -c anaconda tensorflow` .

However, as deep learning is a computational activity, it is highly recommended a user make use of their GPU if they have one. To do this on windows CUDA drivers need to be installed. A full guide on this can be found on the Tensorflow website. When CUDA and CUDNN have been installed successfully then install tensorflow (above version 2.0 is GPU enabled, else install tensorflow-gpu). The GPU that the models have been trained on is a Nvidia GTX 1650 with 4GB of RAM.

Other libraries to install are matplotlib and numpy using `conda install` but use `pip install` to install Sklearn, Pickle, PIL and imutils.

3.2.2 Basic CNN Model

The first implementation of the model was a raw CNN architecture with 2 layers of convolution, this model ran on an earlier version of the dataset with 300 images and achieved very low accuracy in the range of 30%-40% validation accuracy, from these results the CNN was tuned by adjusting hyperparameters such as filter size, batch size, dimension sizes and step size. These changes worked in various combinations and provided information on the workings of each, this information would be used to implement the ResNet model that was chosen from previous research.

```
##Create CNN variable that will contain the network
cnn = tf.keras.models.Sequential()

##Add Convolutional Layer 1
##How many feature detectors
##Size of feature detector 3x3
##Activation Function = ReLu
##image dimensions 200x200x3 colour!
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[200,200,3]))

##Apply MaxPooling -> Size and Stride
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

##Remove input shape as this is only used to connect first layer
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

##Flatten all Previous layers into 1D
cnn.add(tf.keras.layers.Flatten())

##Add fully connected layer
##Number of hidden neurons
cnn.add(tf.keras.layers.Dense(units=128,activation='relu'))

##1 unit and softmax
cnn.add(tf.keras.layers.Dense(units=3,activation='softmax'))
```

Figure 7: The Basic CNN architecture with accuracy ranging from 30%-40% depending on the parameters.

3.2.3 ResNet Implementation

The ResNet works in blocks, this network passes an identity from each block to the next to increase generalisation and combat overfitting. To create a base result 2 blocks of the architecture were implemented. Each block contains Batch Normalisation, the Relu Activation function and a convolutional layer of (3x3) kernel/filter size and a dimension size that varies block to block, these layers are added together. The layers are repeated twice in each block. Next, the input identity is added to the addition of these layers to create a new variable, this variable is used within the next block.

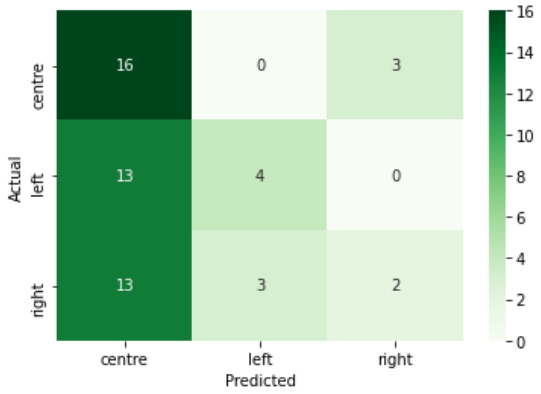
After the final block, the result is passed into a flatten function to create a linear array of probabilities, the result of this is passed through a Relu Activation function and finally passed to a Dense network function that uses the softmax activation which separates the classification categories into their respective output as a fully connected network. The outputs are in an array shape as follows: [1, 0, 0] being centre, [0, 1, 0] being left and [0, 0, 1] being right. These outcomes are in the form of prn obabilities and are rarely a 1 or 0 unless the network had previously trained on the exact image from the input.

The model architecture makes use of the ResNet structure where certain layers are skipped using a connection which contains a (1x1) convolution. This connection links a previous identity passed a residual block of operation to the next block acting like a memory or callback, this helps keep the original identity through the network. With this feature many layers of convolution can be added to a model.

```
# Input
img_input = Input(shape=(img_height, img_width, img_channels))

x1 = Conv2D(8, (5, 5), strides=[2,2], padding='same')(img_input)
x1 = MaxPooling2D(pool_size=(3, 3), strides=[2,2])(x1)

# 1st res block
x2 = BatchNormalization()(x1)
x2 = Activation('relu')(x2)
x2 = Conv2D(8, (3, 3), strides=[2,2], padding='same')(x2)

x2 = BatchNormalization()(x2)
x2 = Activation('relu')(x2)
x2 = Conv2D(8, (3, 3), padding='same')(x2)

x1 = Conv2D(8, (1, 1), strides=[2,2], padding='same')(x1)
#Skip connection
x3 = Add()([x1, x2])

# 2nd res block
x4 = BatchNormalization()(x3)
x4 = Activation('relu')(x4)
x4 = Conv2D(16, (3, 3), strides=[2,2], padding='same')(x4)

x4 = BatchNormalization()(x4)
x4 = Activation('relu')(x4)
x4 = Conv2D(16, (3, 3), padding='same')(x4)

x3 = Conv2D(16, (1, 1), strides=[2,2], padding='same')(x3)
#Skip connection
x5 = Add()([x3, x4])

x = Flatten()(x5)
x = Activation('relu')(x)
```

Figure 8: The first implementation of ResNet using two blocks.

This ResNet performed the same as the basic CNN, with 40.7% accuracy. After close inspection using cross validation matrixes a bias could be seen in the dataset for the centre category.

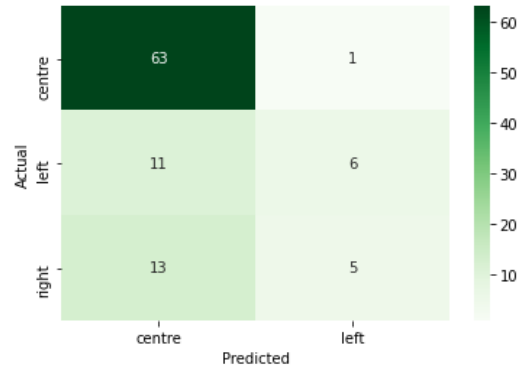



```
test accuracy: 40.74074074074074
```

Figure 9: Cross Validation matrix result showing most predictions as centred with a 40.7% accuracy.

This indicates that if the model was to predict the centre class the majority of the time it would be correct. Therefore, the dataset must be too closely alike to be able to predict or the model is extracting the wrong features.

After manually removing images that seem closer to centre than their respective classes, a greater accuracy was obtained but still a bias on centre class was occurring. In the problems section of this document the dataset issues are discussed more in detail and how better datasets were gathered.




```
test accuracy: 69.6969696969697
```

Figure 10: Cross Validation matrix result showing most predictions as centred with a 69.69% accuracy after a dataset manual cleaning..

At this stage, two more residual blocks were introduced to the network. Each new block increases the filter's dimensions by double per block. This produced greater results and eventually the input image dimension was changed to 250x250 which was the maximum size the model could hold. With this change in parameters and model size an accuracy of 75% was obtained but severe overfitting was occurring, the overfitting will be discussed in the problem section of the document.

```
[INFO] evaluating network...
              precision    recall  f1-score   support

      centre       0.83      0.50      0.62        10
        left       0.88      0.88      0.88         8
       right       0.67      0.86      0.75        14

    accuracy                           0.75        32
   macro avg       0.79      0.74      0.75        32
weighted avg       0.77      0.75      0.74        32
```

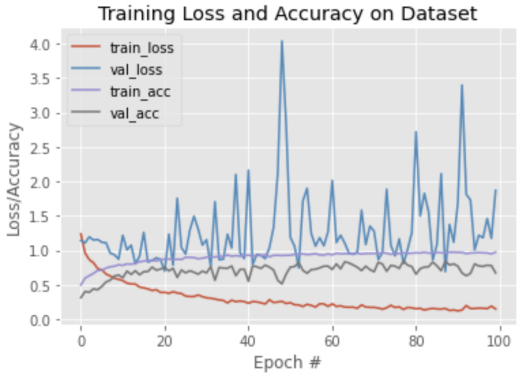


Figure 11: A deeper ResNet model which created high accuracy but high loss.

After more optimization of parameters, the introduction of a dropout function and regularisation implemented on the convolutional layers a more accurate model was produced but there is still a case of overfitting present.

Figure 12: The best model was obtained when training on the largest 54GB dataset gathered, as depicted by wandb graphs..

Weights and biases library called wandb was added to the tensorflow environment on anaconda to more easily access results of model training. This program creates instant accuracy results of the models training in real time as well as current hardware conditions such as memory usage, GPU memory usage, GPU temperature and GPU utilization.

3.2.3.1 Final ResNet Model

The final implementation of the ReNet architecture had a lower range of 312,000 parameters, this model was 12 layers deep but used larger strides in the Convolutional layers as well as smaller dimensions of filters. These changes were to reduce complexity in a way to reduce overfitting that was occurring on more complex models. This overfitting is occurring due to the current dataset not being diverse enough as will be discussed in the challenges section of this document.

```
# 4th res block
x8 = BatchNormalization()(x7)
x8 = Activation('relu')(x8)
x8 = Conv2D(128, (3, 3), strides=[3,3], padding='same',
            kernel_initializer="he_normal",
            kernel_regularizer=regularizers.l2(1e-4))(x8)

x8 = BatchNormalization()(x8)
x8 = Activation('relu')(x8)
x8 = Conv2D(128, (3, 3), padding='same',
            kernel_initializer="he_normal",
            kernel_regularizer=regularizers.l2(1e-4))(x8)

x7 = Conv2D(128, (1, 1), strides=[3,3], padding='same')(x7)
#Skip connection
x9 = Add()([x7, x8])

x = Flatten()(x9)
x = Activation('relu')(x)
x = Dropout(0.5)(x)

# Output probability
steer = Dense(output_dim, activation='softmax')(x)

# Define the model
model = Model(inputs=[img_input], outputs=steer) #add call
```

Figure 13: This is the latest residual block formation of the network

3.2.4 Testing The Model

The evaluation outputs from Tensorflow are useful and a good basis for the effectiveness of the model. However, as this project is a practical application of a CNN architecture, a more practical test will be needed to evaluate the true nature of the model.

To do this a Python script was developed in Jupyter Notebook that allows video playback. This playback is split frame by frame as the video proceeds. When the frames are extracted they are manipulated using OpenCV to create a frame that can be managed by the ResNet model. These predictions are then pooled together using a queue function within Python. The mean result of all the queued predictions is then output into a LabelBinarizer, here the LabelBinarizer labels the prediction mean/average as either centre, left or right based on the maximum probability it has been given.

The output can then be passed to a writer function from OpenCV, this function overlays the output as text onto a copy of the original frame. The frames are then cast with OpenCV, if this operation needs to be aborted, pressing ‘q’ will destroy the operation.

Figure 14: Video classification has predicted to stay straight

Figure 15: Video classification has predicted to turn left

Figure 16:Video classification has predicted to turn right

3.3 Flying

The UAV used for this project was the Parrot Anafi. Parrot also provides an SDK called Olympe which can be used to program functionality to the UAV. The third stage of this project was to implement the best model outcome, discussed in the previous section, into the UAV, the model would then make predictions based on video streaming that came from the UAV.

3.3.1 Tools and Libraries Installation

The first thing that is needed to create a controller to run the Anafi is Ubuntu 18.04 OS specifically. As a result of virtual machines being unable to use nvidia drivers, Ubuntu 18.04 was installed on a MSI Apache GE70 which uses a GTX860M GPU with 2GB of RAM. Once Ubuntu was installed, the Olympe SDK was installed by running the commands as follows:

1. `Cd $home`
2. `Mkdir code/parrot-groundsdk`
3. `Repo init -u` https://github.com/Parrot-Developers/groundsdk-manifest.git
4. `Repo sync`

If problems occur during installation or there are missing libraries, use `pip install` to install the remaining requirements and try running repo sync again..

Next, Olympe will need to build it's virtual environment, once again, if libraries are missing after the build has finished use `pip install.` When everything has successfully installed itself run this shell script to activate the Olympe environment:

```
source ~/code/parrot-groundsdl/./products/olympe/linux/env/shell
```

This environment enables the use of the ground sdk, all available functions and API can be used to interact with the Anafi.

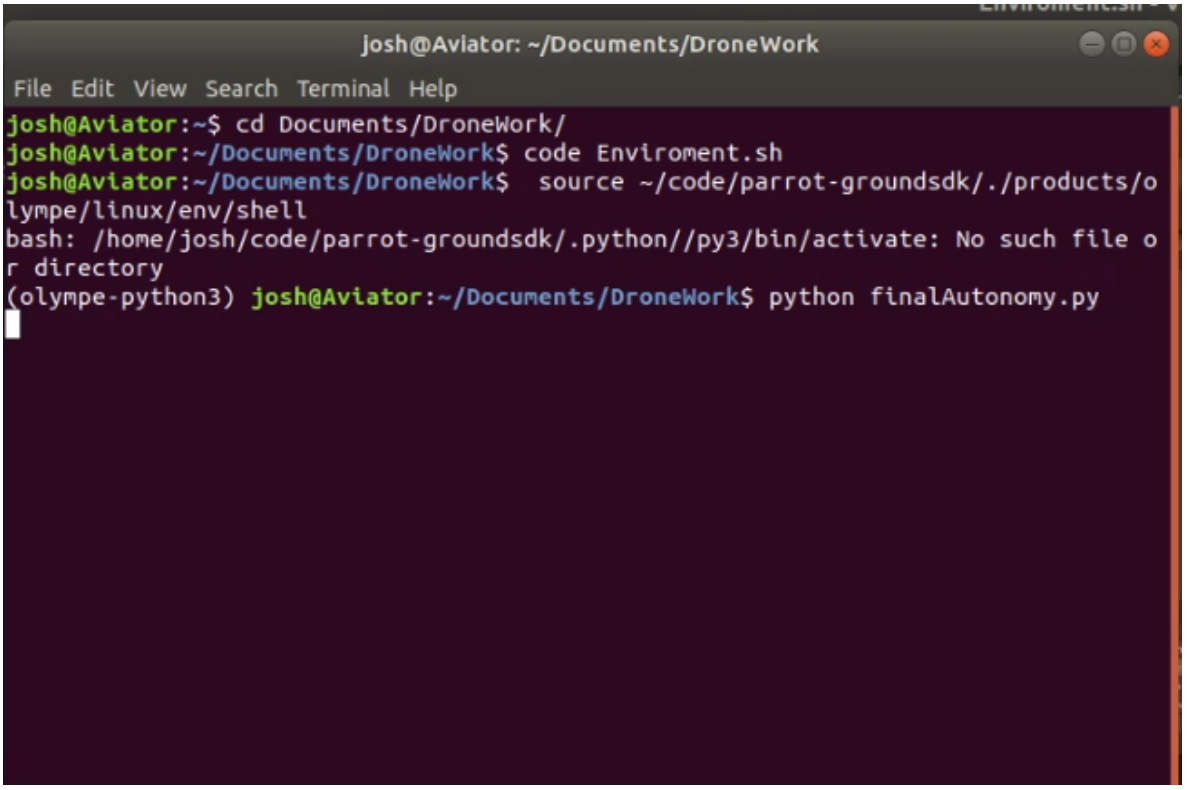


Figure 17: Starting an Olympe Environment in the Ubuntu 18.04 terminal.

As well as the Olympe controller Parrot also offers a Virtual simulator for programming simulated UAVs in a simulated environment. This feature was useful to learn how the API works to control the UAV and what features are available. Doing this made programming the physical UAV much simpler.

Then install the GPU version of Tensorflow, this is very different than before but perhaps easier in Linux. Follow the terminal commands from the Tensorflow website in the Linux section. After tensorflow has downloaded, run the test script provided on the website to make sure the GPU is detected and being used correctly. Finally install other libraries such as Sklearn, Pickle, Imutils and PIL.

3.3.2 Creating a Flight Algorithm

After creating the video classification script, it was crucial to be able to localise individual frames from the UAVs video stream in real time. The UAVs stream is in a raw yuv format, OpenCV is able to convert this format just like the video classification script was done.

In the final version of the flight algorithm an object would be created with threading enabled. Threading is a crucial part in getting the UAV to run smoothly and prevent locking of functions. Each frame will be stopped in the queue and proceed in order while referencing itself after passing through a function to show the yuv frame using OpenCV.

The Olympe environment and API needs a reference of what UAV is to be connected therefore a start function was created where the UAV was connected via the skycontroller IP. Next, the UAVs video streaming is started and the callback functions for streaming are created for live frame processing.

As with the start, a stopping function is required to end the UAVs streaming process and also land the UAV safely. All this can be done through API calls within the stopping function. However, the

stopping function is connected through the video stream and like before pressing ‘q’ will trigger the end of the stream as well as land the UAV safely. A text file called Navigation.txt is also appended to read ‘stop’. This is to signal a flight algorithm that the UAV has stopped. In case of emergency it is recommended that the CTRL-C key combination is pressed in the terminal to abort flying and land the UAV safely.

Each frame from the video stream will be predicted as to the direction to take. To take this direction, the label output from the LabelBinarizer is passed to a text file, this text file can then be read in another function. Using conditional operators the label is said to be either centre, left or right. Based on this, the UAV will move using it’s moveby API function. A slow turning rate is used as not to overshoot the right direction. If the prediction output returns a centre label the UAV continues to move forward.

4 Challenges Encountered

There have been many challenges throughout the lifespan of this project that range from technical, managerial and experience. However, these challenges are what provided interest in the project and the motivation created from solving each has been a driving force in getting through a tough year.

4.1 The Dataset

Acquiring the right dataset for this project is a major key to the project's success and in total over 150GB of data have been collected and put to the trial on model evaluation and video classification. Much of the failures in earlier datasets are due to bad positioning that creates too much likeness in each class or artifacts in the images that the model learns upon and therefore trains and tests well but is faulty in real testing.

In many instances of gathering datasets while driving, the camera setup would vibrate and slowly change in its positioned angle. This led to the edges of the windscreen being visible at the outside of the images or in more severe cases the camera falling off it’s mount completely.

Figure 18: Here a central classification image shows the dash of the car can be seen

As the camera being used had image stabilization enabled sometimes there is no dash seen. Due to these artifacts a lot of the dataset was rendered useless as during a government issued covid lockdown, many parks and walks were not accessible, this left my options extremely limited to my house and laneway.

In the earliest versions of datasets, images were gathered from one specific angle. These images were then manually processed and assigned a classification of centre, left or right. This was a time consuming task and therefore changed to a more efficient system of using different camera angles.

There is also an issue in using different camera angles as multiple trips of the same route should be made to keep the dataset more regular. In doing this many changes occur in the landscape and even personal positioning after returning and setting the camera to a new angle.

4.2 Time

It was an unfortunate turn of events that a pandemic had hit the world while this project was taking place. Many setbacks came from the college's direction in examinations being more curriculum assessment driven, thus much time had to be split into completing projects with earlier deadlines while also maintaining momentum with this project. A later starting date to the college year was the cause of a slow momentum in entering the year, also induced by the confusion of events taking place worldwide.

Given these dramatic changes to the normal year, this project is definitely a far more time consuming project rather than creating an app or website due to it’s Machine Learning nature. Most of the later stage models that were trained took more than 10 hours to complete. Even so the results of the models are not always to be a success and many end in time being wasted. In tandem to this the laptop that runs the models is my personal laptop that has been used throughout the year for virtual college attendance and work, as a result of resource exhaustion, many unexpected shutdowns occurred.

Gathering the datasets was also a time consuming activity. The majority of the project taking place over winter during dark evenings and afternoons reduced progress by a large margin.

4.3 Technical Issues

Out of Memory (OOM) was an ongoing scenario throughout this project. This was caused by overloading the memory available for the GPU. OOM can happen in a couple of different ways such as the input image being too big, individual parameter dimensions being too large and training and testing batch sizes being too large.

The Olympe installation took many attempts to perfect and entails the user having to manually install the correct libraries that are missing using `pip install` . Due to a small community on the Parrot forums with what seems as one developer answering questions there, troubleshooting issues with the UAV and Parrot software were either tedious or a waste of time.

As with the Olympe software installation, Tensorflow was as tedious to install for GPU usage. If the wrong CUDA toolkit or CUDNN has been installed, the GPU will not activate.

4.4 Overfitting

Overfitting is caused when a model begins to memorize training data rather than training on the data, as a result, the model will perform much worse on test data.

```
[INFO] evaluating network...
              precision    recall  f1-score   support

      centre       0.75      0.25      0.38        12
        left       0.88      0.58      0.70        12
       right       0.40      1.00      0.57         8

    accuracy                           0.56        32
   macro avg       0.67      0.61      0.55        32
weighted avg       0.71      0.56      0.55        32
```

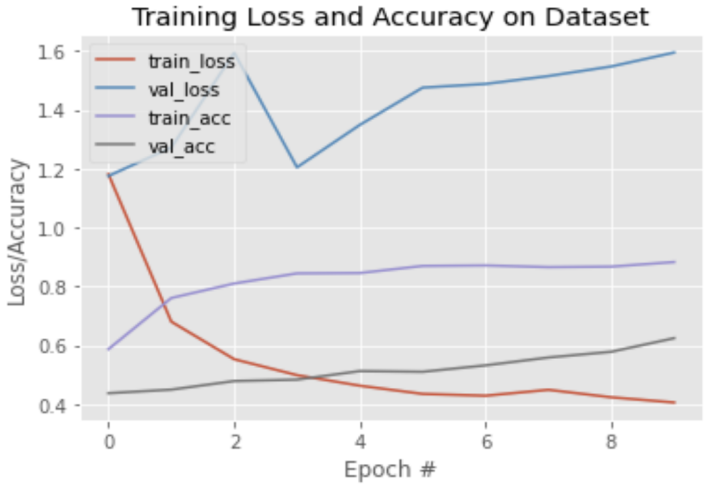


Figure 19: An evaluation from an early ResNet model, as can be seen there is a high training accuracy with an extremely low validation accuracy, this is an example of a type of overfitting.

Overfitting was addressed throughout the project by changing the dataset, introducing Dropout, introducing Regularization and reducing hyperparameters. This challenge is still affecting the training of current models to a lesser degree. In the latest case of overfitting the model is producing very high results for training and validation but does not perform as well in practise.

This is due to the current dataset not having enough variety, this could have been addressed if travel was available during lockdown but unfortunately there are no local walks or tracks within my area and all current data has been gathered on my private property. Many varieties of model were implemented upon the dataset with all results varying but concluding in the same type of overfitting.

Figure 20: Latest evaluation showing overfitting .

4.5 Laptop Battery

When testing my models on the UAV, I needed to travel to the right locations. The Olympe controller could only be installed on an older laptop. As a result, when operating the Olympe controller on this laptop, the laptop would run out of battery within 20 minutes. Therefore, only one test flight could be tested by the time I had gotten to the location. This was an unavoidable challenge as Ubuntu could not be installed on the more modern Dell laptop as it uses a PCI SSD, the bios on the Dell would block installations on the drive which resulted in not being able to dual booth and wiping the Windows OS was not an option.

5 Learning Outcomes

Every factor of this project has been original and enjoyable, errors being exempt from this statement. The technologies used were the latest in machine learning/deep learning and are truly the cutting edge of technology as they can process so much information contained in an image, all being processed on home laptops one of which is several years old.

5.1 Tensorflow and Keras

Machine learning/deep learning have been the heart of this project and libraries like Tensorflow, which has enveloped Keras now, have made an extremely complicated subject much more manageable and usable. After working with the library for so long, it has bolstered my personal experience, in a field I was inept in, ten fold. I would be confident in my ability to use this library in the workplace after this project.

5.2 Python

Python was introduced to me last year, and while it was a good introduction, the true power of Python had not been breached. Python was the driving force language when implementing this project and was the perfect coupling to the technologies available and libraries I would use. Threading in Python has been introduced over the course of this project and really helped smooth the running of the UAVs video streaming. I enjoyed Python so much that it is now a target to meet when looking for employment.

5.3 Machine Learning/Deep Learning

The machine learning that has been used in this project is some of the latest techniques in the industry and is used widely in the industry. Currently deep learning and specifically image classification has been gaining in popularity after such companies like Volkswagen and Tesla have been introducing more autonomy in their cars. Data production and consumption are at all time highs and there are many positions available in the industry looking for data analysis.

Over the course of this project I have become capable of deducing problems that are occurring in the training of models, how to best optimize a model architecture to produce faster results while also maintaining the models accuracy. To produce effective results I've learned how to augment image data and pipeline it to the model.

There are also many mathematical functions that are incorporated in machine learning, all of which have unique effects on model learning and accuracy. I've gained an understanding of such functions as: Relu, Softmax, Batch Normalization and categorical cross entropy.These functions help to create an optimal model, while functions such as: Adam, and the regularizers effect on it. A function which creates a careful search for optimal results.

The models I created were then implemented into a real world scenario to fly a UAV much like an autonomous car. The UAV however, could only assess it's direction based on only camera streaming and no other telemetry devices.

5.4 Time Management

In order to complete the project, time had to be managed very carefully as each iteration of the model created would take up vast amounts of time. To increase the efficiency of time management I was lucky enough to own a second laptop where I could code UAV movement and test how the UAV would pair with different models.

I used an Agile methodology to continuously develop, if I could not complete an implementation in the time I had given myself, that implementation would be then put on hold until the next implementation could be finished and tested.

6 Achievements

Overall the project varies in success. There have been many variables in this project that breakdown other variables in a domino effect. The main leech on the achievements gained has been the dataset. However, The implementation of the model trained on this not so good dataset, has successfully controlled the UAV around corners outside my house and the UAV has maintained course on straight sections of track.

Getting the UAV to move while also processing video was a major achievement. The UAV should have been able to move asynchronously while video streaming. However the introduction of our model when calling frames to a thread had a bad effect on this asynchronous movement. The solution of separating both to be working on their independent queue of actions was to have the queue processing images to output predictions to a text file. This text file would then be read in a separate function where the move API could then be implemented successfully.

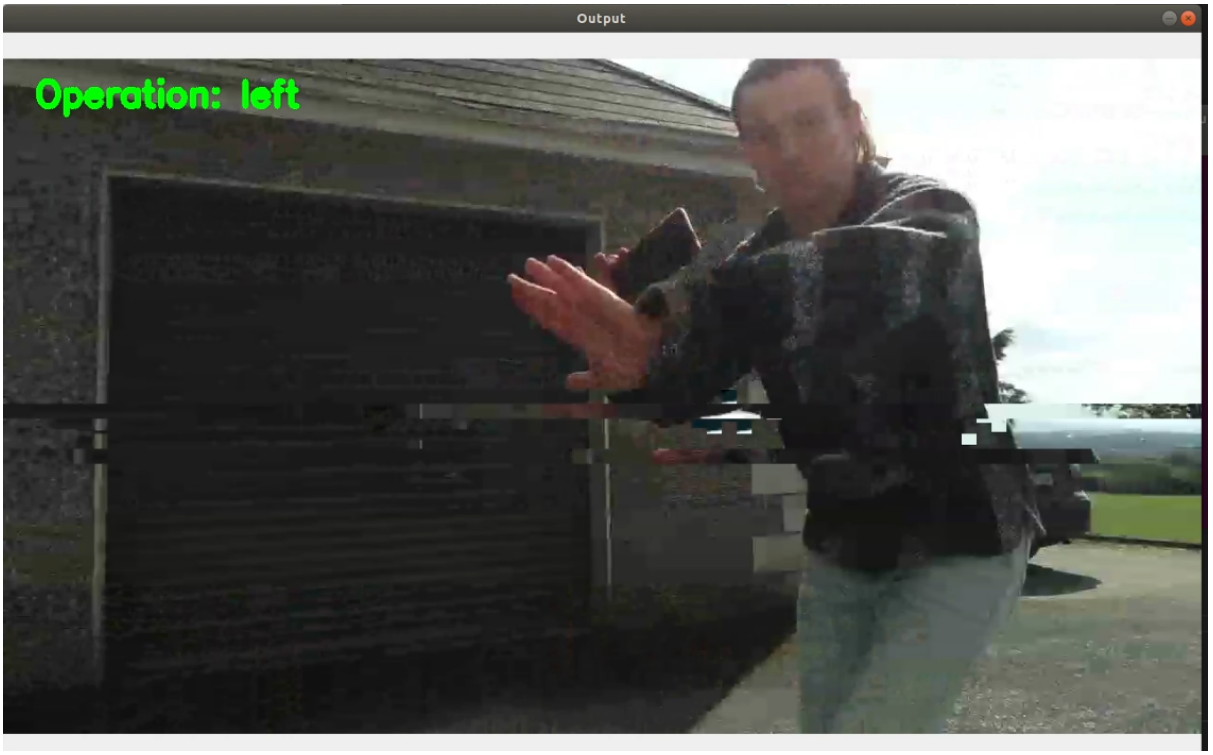


Figure 21. Are you alright? No, I'm all left!

7 Improvements and Future Overhauls

Although the project has been successful in what was set out from the beginning, there is room for improvements in all areas of the project. Adding features was out of the question during the timeline provided but if the project was to be picked up again some features and improvements would include producing a dataset with far more variety, adding obstacle avoidance and then implementing it to the UAV.

7.1 Better Dataset

The greatest issue during the project was the dataset. It is the golden goose of the project's outcome and creating a viable dataset was severely overlooked in my research.

One change I have already discussed in section 4.1 would be to acquire multiple cameras to gather the dataset in one run of a given route. This would provide a much more stable comparison of classifications for the model to learn upon.

More of the right kind of data would be acquired from lots of different locations. This was impeded by the national lockdown. If more diverse data was collected, the current overfitting issue could be solved and a far more versatile and stable model could be created and implemented on the UAV.

Positioning of cameras while driving was a huge problem in the gathering of a viable dataset. Much of the dataset collected when driving was made obsolete by artifacts in the image such as rain and dirt on the windscreen and the dash being visible in images. Much better results could be obtained if the camera was mounted outside the vehicle or inside on the roof, looking down upon the road but overlooking the dash and bonnet of the car.

7.2 Obstacle Avoidance

To really enhance the UAVs navigational prowess, obstacle avoidance would be a fantastic feature to implement. If the UAV could move around an obstacle and stay on a route it would be quite impressive and improve the overall usability of the features already implemented in this project.

There are a number of ways this could be done but one that stands out the most as being easier to implement, would be to have the model make a separate prediction in which objects have a chance of impeding trajectory. Another class would be made to learn from that would show objects close to the UAV, such as cars, people, lampposts and other random objects. The UAV would then make a prediction on whether it will impede its navigation. If it is over a certain threshold the UAVwould rotate to a safer trajectory, however, if it cannot do so, the UAV would stop.

7.3 Tensorflow Pipeline

Using a Tensorflow pipeline instead of the Keras flow from directory would have increased the learning rate of the dataset. Keras flow from directory does not make use of the GPU and relies on the CPU to import images to the pipeline. With Tensorflow’s data pipeline, tensors are used to traffic image data to the model, these tensors run on the GPU and are much faster than the CPU at producing a pipeline. It can be seen when GPU utilization is increasing using the wandb library discussed earlier.

7.4 GUI Implementation

Providing the user with a GUI was not essential as this project was a proof of concept rather than consumer driven application. With this said, a GUI would be nice to combine the features 0of the project in one space.

If a GUI was to return the video stream to the user where the user can use buttons on screen to start flight or stop flight, this would greatly add overall usability. The GUI could also return location information using the UAVs GPS feature, this could then be displayed on screen.

8 Conclusion

To conclude this document, the project has been fruitful in both learning outcomes and achievements. The model implementation has worked successfully on many datasets of varying quality. These models were successfully evaluated and changed to better be optimised. Even though the datasets gathered were not perfect, the models trained upon them were able to navigate the trajectory on test videos.

Ultimately these models were incorporated into a flight algorithm developed for the UAV. This algorithm received predictions from the UAVs video stream and judging on the prediction a suitable direction was relayed to the UAV, the UAV then navigated to the optimal direction based on these predictions.

A video output was displayed to control the laptop. This stream had minimal delay and provided the current operation predicted visibly on screen. The operation could be abandoned by pressing the ‘q’ button while on the video stream was up. This action closed the video stream and safely landed the UAV. If the UAV did not respond to this command, a redundancy was implemented by pressing ‘CTRL-C’ in the terminal the stream would shutdown and the UAV would safely land.

9 Acknowledgements

I would like to express my sincerest gratitude to the following people for all their help and guidance through this project without these people this project would not have been possible.

- Dr. Oisin Cawley
- Dr. Joseph Kehoe

10 Plagiarism Declaration

Declaration

- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offense.

Student Name: Josh Hudziak

Student Number: C00231846

Signature: Josh Hudziak

Date: 30-04-21