1. Introduction

Modern developments in machine learning and autonomous vehicles have allowed research towards automatic navigation systems for different environment types to become much more feasible and widespread. For example, the rapid emergence of drones in the aviation industry [1] have made their general use much more viable, and the development and wide-spread use of machine learning algorithms like deep neural networks has grown in the last decade, and is able to overcome the limitations of older algorithms that rely on hand-crafted features [2].

Following man-made paths as a means of navigations has been a topic of growing interest. Such is the case with self-driving cars; tracking and navigating paved roads needs to be done accurately to avoid fatal accidents involving humans, and has been the focus of much research in the last few decades, with major progress having been achieved in particular in the past few years [3]. There is value in researching means to automatically follow other kinds of man-made paths, such as hiking trails. For instance, large counts of SAR (search and rescue) incidents occur during hiking; in the US alone there has been 78,488 individuals involved in SAR incidents from 1992 to 2007, with hiking being the most common cause (48%) [4]. A robust trail following algorithm could have applications in SAR missions lead by mobile robots, or as a tool to assist humans in navigating through difficult trails, including people with handicaps that would make traversing trails difficult without aid.

Finding a generalized solution to hiking trail navigation is a problem that is still mostly unsolved. One of the main reasons is that hiking trails have much greater degrees of variability than paved roads, the latter having made great strides in creating robust, fast, and generalized algorithms to detect and navigate them [3]. For example, a large bumpy dirt path is vastly different from a desire path consisting of crushed grass, whether that be in size, in composition, or in the subtlety of the trail. Sometimes there might not be a direct path to follow, and instead the decision must be made through other contextual clues, such as by identifying an opening in foliage in the distance. Such cases would lead to the failure of conventional path detection algorithms that try to detect features of the path itself. Depending on the environment, large amounts of noise can also be produced, such as from the density of surrounding vegetation and the shadows they cast.

State-of-the-art approaches to this task have included the use of hand-crafted features, segmenting the path from its surroundings. An example of such an approach uses appearance contrast [5] which creates a trail region hypothesis based on a learned distribution of trail curvature and width, assuming the trail region to be of triangular shape. Another approach utilizes visual saliency [6], which segments the path from the background by first quantifying

how much each pixel stand out from the rest in an image; pixels on the path are labeled as high-saliency as opposed to the background, and a path segmentation can be computed from the resulting map.

Machine learning approaches have been used to solve similar problems. These algorithms are suited to create more generalized solutions without the use of hand-crafted features, which is an important prospect for a task where the characteristics of a trail can greatly vary. One of these approaches trained a deep neural network for a biomedical image task; segmenting neuron membranes by classifying each pixel of an image as either a “membrane” or a “non-membrane” pixel [7]. This kind of solution developed for the noise-rich environments of biomedical images demonstrates the feasibility of machine learning solutions for following paths in highly noisy images, the principle of which can be applied to hiking trails. Another approach more specified to our task trained a convolutional neural network to select turning directions based on image data to follow a hiking trail, with enough accuracy to allow the automatic steering of a low altitude drone on a trail [8].

This paper aims to develop a machine learning approach for hiking trail direction perception. While machine learning solutions tend to be computationally expensive, the aim is to minimize that cost to make a model usable in low resource platforms.

2. Approach

2.1 The data

Hiking trails can be found in many different types of environments such as rock-strewn mountains, forests, and bays. Ideally collecting data from as many types of environments as possible would lead to better generalization, however our data will be limited to being recorded from trails with similar foliage rich environments.

Navigational algorithms can use various kinds of data, from laser sensor data to images. This paper limits itself to a computer vision approach. This has the advantage of allowing the model to be easily implementable in systems for human use, namely through smartphone cameras, while also being applicable for mobile robot navigation. Moreover, there is no dependence to GPS, as it cannot be assumed to be available or accurate in forests [9], where it is expected for the model to be consistently functional.

Entry images are labelled with 1 of 3 directional categories, based on the direction of the hiking trail in relation to the picture: “left”, “right” or “straight”. This labelling approach was chosen over labels representing angles in order to simplify the task to one of classification instead of regression, and to simplify the dataset building process.

The dataset is available at:
https://www.kaggle.com/dataset/c89aa7511b75aeb9b8a2ed35b43f449d35ad7b788d645a57153c14e0b52937d1.

2.2 The model

Many possible computer vision classifying algorithms can be considered. Because of the factors mentioned previously regarding the high variability of trails appearances, a deep neural network is a suitable choice to avoid assuming/defining concrete key features to be extracted from images. This neural network will have for input only the raw pixel values of the image, outputting prediction confidence values for 3 possible direction classes (“left”, “right” and “straight”). It would then be possible to either select the direction with the most confidence, or to estimate the directional angle to follow the trail by calculating an average of the confidence values for all three directions.

Because of this being a computer vision task, a convolutional neural network classifier is implemented; its matrix-based operations making it suitable for image classifying [10]. A recurrent neural network is unfit as the data is treated as individual images and not video; instead of making a network that can track a hiking trail over time, the objective is a model that can guess the direction of individual images. The neural network outputs three directional confidence values, the highest value can then be selected as the direction to follow.

Neural networks tend to be computationally expensive. This is a drawback that should be minimized so that the model can be deployed on low resource platforms. The MobileNetV2 architecture is selected, a convolutional neural network architecture which trades some accuracy to vastly decrease the number of model parameters and operations it performs, with state-of-the-art performances on tasks such as image classification [11].

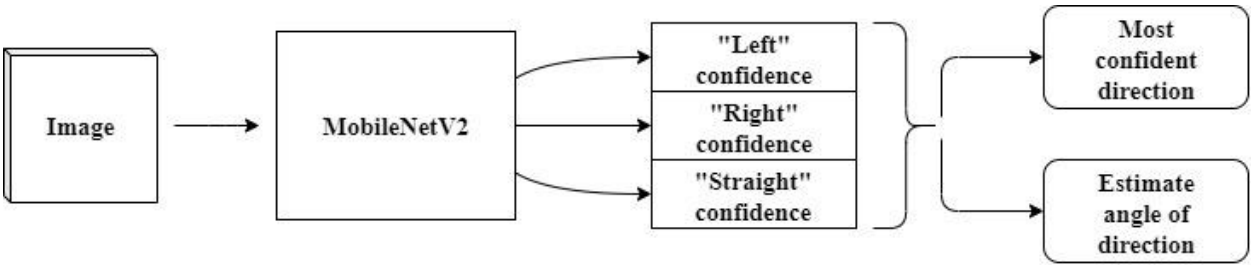


Figure 2.1: Representation of the full path direction detection system. An image is fed into a trained MobileNetV2 neural network and outputs three directional confidence values, the maximum of which can be selected as the correct direction, or a directional angle can be estimated from an average of the three confidence values.

The MobileNetV2 architecture uses depthwise separable convolution, which consists of a depthwise convolution followed by a pointwise convolution to greatly decrease the number of operations performed when compared to standard convolution layers.

Residual connections are added between low-dimensional tensors that surround depthwise separable convolutions, which allow for higher model accuracy and faster training without compromising the architecture's aim of minimizing the size and computational complexity. Linear bottlenecks are added after these connections; the use of a linear activation function is selected over a non-linear activation function as to reduce information loss.

3. Building the dataset

3.1. Gathering footage

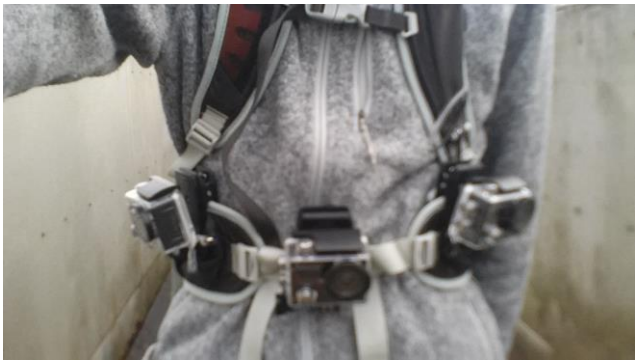


Figure 3.1: Picture of the recording setup taken during a recording session. The recording setup is composed of 3 cameras (Campark ACT74 Action Camera, Amazon provider) that is attached around the waist: a camera at the center to capture footage labelled as the “straight” directional class, and the other two angled 45° to its sides to capture the “left” and “right” classes respectively, covering a full range of around 180°. A hiking backpack is used to attach the cameras around the waist, by attaching clips on the waist strap. The cameras are set at a height of around 0.96 meters from ground level. Footage is recorded at a resolution of 1280 x 720 pixels, with a frame rate of 60 frames per second, lens angles of 170° and cameras angled slightly downwards (around 15°, however natural shifts of posture while walking makes that angle very imprecise).

While recording, walking is done at a constant rate with a moderate walking speed (around 3mph), standing straight and keeping the arms behind the back to avoid obstructing the cameras. This setup allows for equal amount of footage to be gathered for all 3 classes, avoiding class imbalance. Recording at 60 frames per second is done to gather less blurry images than if recording at a lower frame rate. Motion blur would be an issue otherwise, especially because of the natural wobbling created when walking.

Two distinct locations are selected to record the footage: Southampton Common and New Forest (Ashurst). While New Forest contains a more traditional forest-like hiking trail, Southampton Common contains a higher variety of path types, such as large sand paths, paved paths, grassy desire paths and small dirt paths surrounded by dense vegetation, so as to create a more diverse dataset.

Trip	Date	Conditions	Video size
Southampton Common (Trip A)	14/11/19 15:04-16:27	Cloudy. Low luminosity from around 16:15 onwards.	4-4.5 GB of video per camera.
Southampton Common (Trip B)	04/12/19 13:15-13:55	Very sunny. Some sun glares.	2-2.5 GB of video per camera.
New Forest (Ashurst)	05/12/19 13:20-14:15	Somewhat cloudy.	5.5-6 GB of video per camera.

Table 3.1: Trips undertaken to gather footage for the dataset. Two recording sessions were undertaken at Southampton Common with different weather conditions, and one session was undertaken at New Forest (Ashurst) for terrain variety.

Only 1 out of every 12 frames from our gathered footage is used for the dataset. All 60 images per second would be excessive, as the differences between neighboring frames is negligible, especially seeing as the recording process is done at walking pace. Little information is lost while saving a large amount of storage space. Before any data augmentation or cleaning, the dataset has a total of 72k entries.

Figure 3.2: Examples of images from the dataset. These images are labelled as “right”, “straight” and “left” respectively, representing the direction to turn to follow the trail.

It must be noted that this footage is recorded in Autumn and that rain is frequent, meaning that fallen leaves are abundant, the ground is often muddy and has puddles. Moreover, the two trails that were recorded from are from the same region and have similar features. This dataset will be biased towards this kind of environment, so a model trained on it would likely show poorer accuracy when applied to other terrain types.

3.2 Including another dataset

In addition to the dataset we are building, another dataset is used [12] to improve data variety and model generalization. For the sake of clarity, we refer to our built dataset as dataset A, and this other dataset as dataset B.

Dataset B differs from our own in terms of environment and the parameters of the recordings; while our footage was recorded with cameras at waist level with a fixed downward tilt, footage for dataset B were recorded with head mounted cameras which adds a lot of data variety in regards to recording height and angle variations.

By keeping both datasets separated, two “partial” models can be trained, one for each datasets. It is then possible to compare their performances to that of a final model that is trained on both datasets. This allows to compare how performance is improved based on data variety, and to make sure the datasets are not biased based on environments or the way the footage was recorded. For example, a model that is only trained and tested on dataset A might show deceptively accurate final testing results, while failing when tested on dataset B; something which would indicate some flaw in dataset A that leads to the model learning features that do not generalize well to other data.

3.3 Data pre-processing

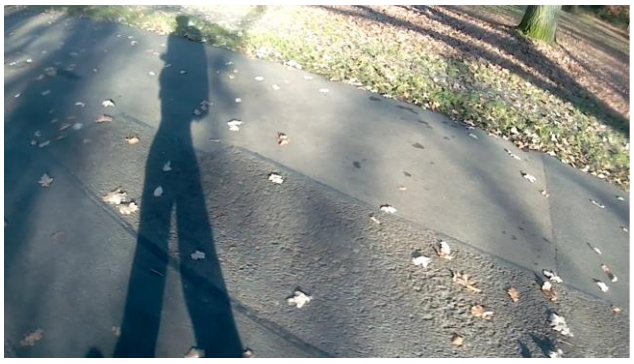
3.3.1 Data cleaning

Depending on our position relative to the sun, our shadow can be cast in front of us or to our sides and are captured by the cameras. Left unchecked, it could lead a trained model to identify these shadows as a feature; guessing a direction based on the shape of our silhouette. Recordings were performed on slightly cloudy days to avoid this issue; light being dispersed enough to not cast a clear shadow. Because of this, not many offending frames were present in the final recordings and it was possible to manually find and remove remaining ones.

Although too infrequent to meaningfully impact the robustness of the data, cases where elements such as arms, knees or shoes are mistakenly present in the frame are removed, whether be it because of vegetation requiring to be pushed aside or crouched under, or moments where we fail to follow the recording procedure of keeping our arms behind our back or to walk completely straight.

Low luminosity is another issue to consider since some of our footage was recorded near twilight. Frames where the luminosity levels are too low have excessive noise, and so are removed. Dark images where there is enough light to make a meaningful guess are kept, as it would be valuable for a trained model to be able to make guesses in low luminosity environments.

In certain cases, there is not a correct direction to follow the trail onward: namely dead ends and intersections. Dead ends are excluded from our dataset, this includes situations in which the hiker does not know how to proceed. However, it would be valuable for a trained model to be able to choose one path to follow when confronted with an intersection; as such, these are included in our dataset, while acknowledging that the directions chosen have a degree of arbitrariness depending on the hiker.



(a) Example of an image removed from the dataset for containing the hiker's shadow. In this example, the direction of the trail can be guessed to be to the left based solely on the shadow, which could become an erroneous feature in a trained model.

(b) Example of an image removed from the dataset, showing the hiker's right arm. In this example, the direction of the trail can be guessed to be to the left based solely on that arm, which could become an erroneous feature in a trained model.

(c) Example of a low luminosity image that is kept in the dataset. The direction of the source of light is the same as the direction of the trail, to the right. Being able to estimate the direction through this kind of information is valuable.

(d) Example of an image containing an intersection that is kept in the dataset. Although there is no definite answer as to what the direction should be, we believe an algorithm should confidently estimate it to be “straight”, corresponding to the decision made by the human hiker in that situation.

Figure 3.3: Examples of images that are removed or kept in the dataset.

Over-representation of paved roads in our dataset is another issue. A large amount of our footage comes from Southampton Commons, which mainly contains paved roads. The goal in the creation of the dataset is to represent hiking trails, which tend to be unpaved most of the time. As such, most paved roads are removed from the dataset. We do not exclude them entirely as there is value for our model to account for them for the sake of generalization.

3.3.2 Data augmentation

Horizontal flips are applied to all images of the datasets, doubling the number of entries. Direction labels are changed accordingly: “left” becoming “right”, “right” becoming “left”, and “straight” remaining “straight”.

Further data augmentation techniques are applied in-memory instead of being stored on disk to avoid use of excessive disk storage space and to also allow easier parameter tweaking. This includes width, height and zoom shifts.

Small rotations are also added to images (in a range between 15° to the left and to the right). This allows the trained model to become somewhat tilt-proof, which is valuable as images cannot be expected to always be captured straight in practical use cases, such as through a mobile robot’s camera or pictures taken by a human. Moreover, the setup used to record footage for dataset A produces images with tilts from the right and left cameras when in motion, which could lead to a trained model falsely recognizing a tilted environment as a “right” or “left” feature, such as through detection of the diagonal edges of titled trees in the background. This allows to mitigate that potential issue.

The Keras [13] *ImageGenerator* class is used to apply random combinations of data augmentations to images in-memory for both the training and validation sets during training.

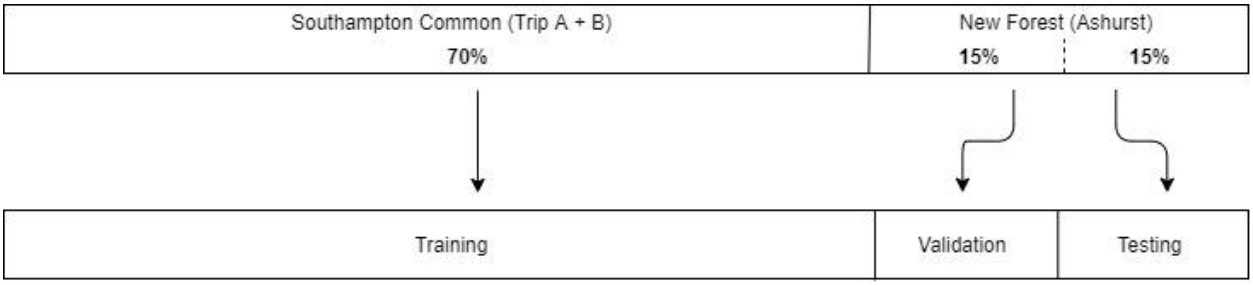
3.3.3 Color space

For color space, we select RGB instead of greyscale. The information loss from using greyscale would decrease accuracy while improving the model’s speed, however the architecture we have selected already has fast performance that make the trade-off unnecessary. Regardless of the color space, we normalize our images, changing pixel values of all channels from its original range of [0, 255] to [0, 1].

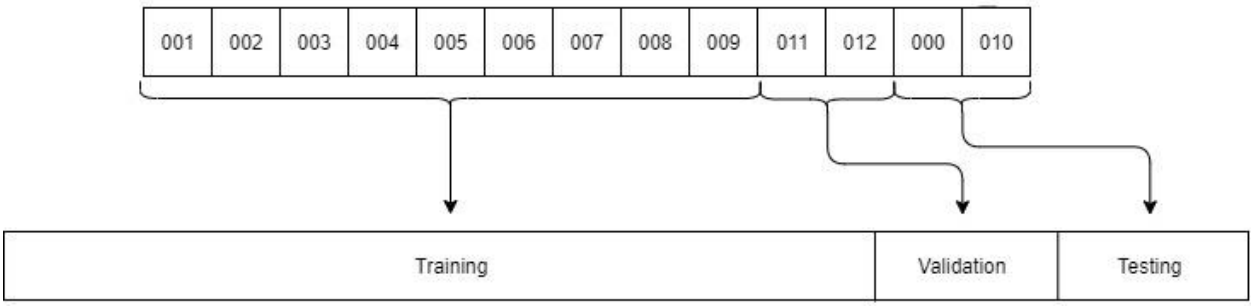
3.4 Training, validation, and testing sets

Our full dataset is composed of two different datasets: dataset A containing 100k entries after pre-processing, and dataset B containing 84k entries. Both are separately partitioned into training, validation, and testing sets. Training and validation sets are used during training phases, while the testing sets are reserved solely for final model evaluation.

Dataset A is composed of two different trails: Southampton Common and New Forest (Ashurst). To avoid bias, we separate training and validation/testing based on the hiking trail. For dataset B, we partition it by folders.



(a) Partitioning of dataset A into training, validation and testing sets, as a 70:15:15 split. Southampton Common entries are used for the training set, while New Forest entries are partitioned further in two, half to be used for validation and half for testing. The 15:15 split for the New Forest data is done manually to avoid having the same trail section appearing in both the validation and testing sets, instead of using a method such as putting every odd frame in validation and every even frame in testing, which would lead to the data of the two sets being much too similar. We avoid a random distribution for the same reason.



(b) Partitioning of dataset B into training, validation and testing sets, as a 79:14:7 split. Folders 001 to 009 are used as training data. Folders 011 and 012 are used for validation data. Folders 000 and 010 are used for testing.

Figure 3.4: Training, validation and testing sets partitioning for datasets.

4. Building the models

4.1 Neural network implementation

An implementation of the MobileNetV2 neural network provided by the Keras library [13] is used. The input tensor is set to be of size 128 × 128 × 3; representing input images resized to be of size 128 × 128 pixels with 3 color channels (RGB). A larger input tensor such as 224 × 224 × 3 was not selected because of the aim of maximizing performance. A dense output layer of size 3 is set, for the 3 directional classes to predict. Weights are initialized randomly.

The Adam optimization algorithm [14] is applied for training, a state-of-the-art stochastic gradient descent optimizer for deep neural networks that can optimize individual learning rates for each model parameter and is able to handle problems with high level of noise.

Because of this being a multi-class classification task, we use categorical cross-entropy as the loss function as it outputs confidence values between 0 and 1 for each direction class.

The width multiplier (named “alpha” in the Keras implementation) is a MobileNetV2 hyperparameter which can be adjusted to increase or decrease the number of filters in each layer of the network, depending on the accuracy/performance trade-off that is desired. An “alpha” value of 1 is selected, the same default width set for the principal network used in the MobileNetV2 paper [11].

Overall, this network contains 2.2 million parameters and performs predictions with an average running time of 109 milliseconds on an Intel Core i5-7200U (4 x 2.50GHz).

4.2 Training models

Three different models are trained on different configurations of our partitioned datasets (see Section 3.4), to compare varying results.

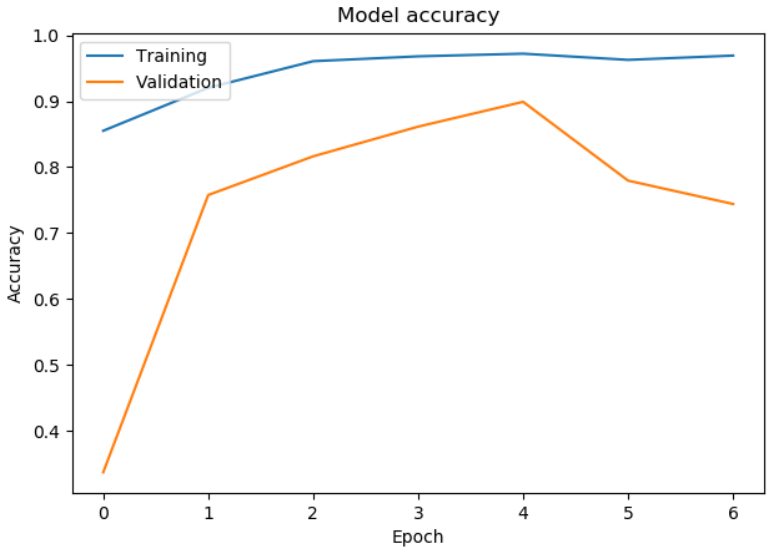
Model Name	Training Data	Validation Data	Testing Data
mobileNet2_1.0_128_A	A	A	A, B
mobileNet2_1.0_128_B	B	B	A, B
mobileNet2_1.0_128_mix	A, B	A, B	A, B

Table 4.1: Training, validation, and testing set configurations to train 3 different models. Model names are taken from their saved file names. Training, validation and testing collumns represent from which dataset the data is partitioned from, whether that be dataset A, B or both. All models are evaluated on the same testing set partitioned from dataset A and B.

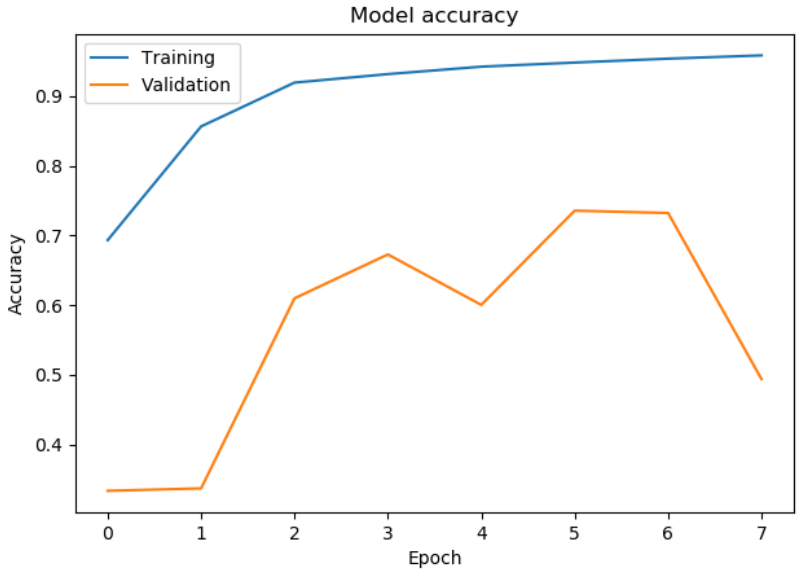
An early stopping function is applied during training; model training stops when the average validation loss at the end of one epoch has not decreased for more than two epochs. This is implemented through the Keras EarlyStopping class.

Because of the large size of our dataset and to decrease training time, batches are used for training, validation, and testing. The higher batch sizes are, the lower a model's generalization becomes [15], however training time greatly decreases but computational cost increases [16]. Because of limited computational resources available for training, a batch size of 32 is selected.

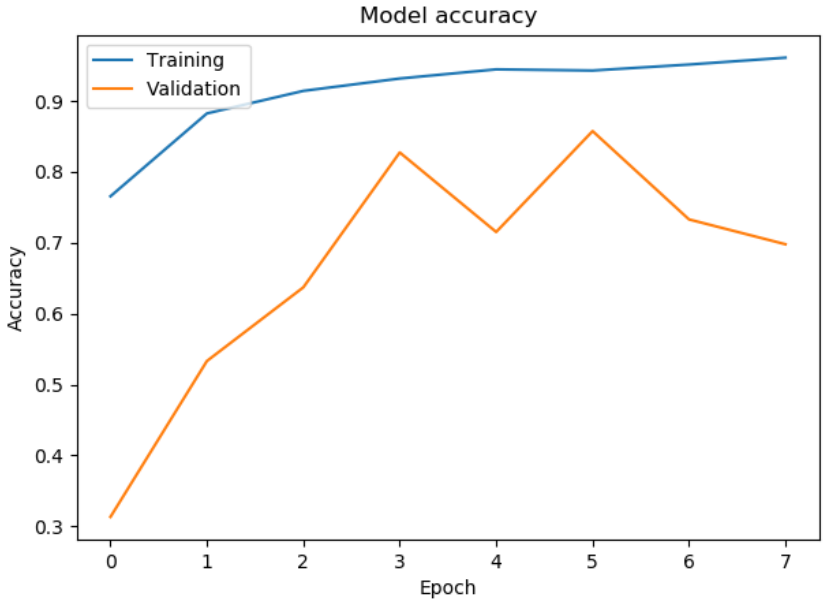
In an epoch, each entry is iterated over, with a randomly data augmented entry always used instead of the original (see Section 3.3.2) to avoid overfitting.



(a) Training history for the model "mobileNet2_1.0_128_A", trained on dataset A. The saved model is the one from epoch 4, with the maximum validation accuracy of 0.899.



(b) Training history for the model “mobileNet2_1.0_128_B”, trained on dataset B. The saved model is the one from epoch 5, with the maximum validation accuracy of 0.755.



(c) Training history for the model “mobileNet2_1.0_128_mix”, trained on dataset A and B. The saved model is the one from epoch 5, with the maximum validation accuracy of 0.857.

Figure 4.1: Training history graphs of the three models. Full history tables can be found in Appendix D.

4.3 Evaluating models

All three models are evaluated on the same testing set that was partitioned from both dataset A and B.

Accuracy is not a meaningful enough metric for final model evaluations; other metrics must be calculated. Confusion matrices are computed (see Appendix E) using the sklearn library's [16] *confusion_matrix* function. Precision, recall and F1-scores are also computed for each class against the others, using the sklearn's *classification_report* function.

For any given class, precision represents the fraction of true positives amongst all positives, recall represents the fraction of successfully labelled entries, and the F1-score is a harmonious mean of the two. Precision is less important than recall for this task, as making no mistakes is less important than being right overall in most circumstances when using this model.

	Precision	**Recall**	**F1-score**
Left	0.85 ; 0.29 (**0.57**)	0.86 ; 0.47 (**0.665**)	0.85; 0.35 (**0.60**)
Right	0.91 ; 0.40 (**0.65**)	0.83 ; 0.41 **(0.62**)	0.87 ; 0.40 (**0.635**)
Straight	0.76 ; 0.41 (**0.585**)	0.85 ; 0.14 (**0.495**)	0.80 ; 0.21 (**0.505**)

Table 4.2: Classification report of model "mobileNet2_1.0_128_A". In each cell, the values from left to right represent respectively: the result from the testing set in dataset A, in dataset B, and the mean.

	Precision	**Recall**	**F1-score**
Left	0.65 ; 0.95 (**0.80**)	0.71 ; 0.59 (**0.65**)	0.68 ; 0.72 (**0.70**)
Right	0.73 ; 0.96 (**0.845**)	0.56 ; 0.86 (**0.71**)	0.63 ; 0.91 (**0.77**)
Straight	0.66 ; 0.67 (**0.665**)	0.85 ; 0.99 (**0.92**)	0.75 ; 0.80 (0.**775**)

Table 4.3: Classification report of model "mobileNet2_1.0_128_B". In each cell, the values from left to right represent respectively: the result from the testing set in dataset A, in dataset B, and the mean.

	Precision	Recall	F1-score
Left	0.95 ; 0.97 (**0.96**)	0.82 ; 0.76 (**0.79**)	0.88 ; 0.85 (**0.865**)
Right	0.98 ; 0.94 (**0.96**)	0.82 ; 0.93 (**0.865**)	0.89 ; 0.93 (**0.91**)
Straight	0.64 ; 0.82 (**0.73**)	0.97 ; 0.90 (**0.935**)	0.77 ; 0.90 (**0.835**)

Table 4.4: Classification report of model "mobileNet2_1.0_128_mix". In each cell, the values from left to right represent respectively: the result from the testing set in dataset A, in dataset B, and the mean.

The model trained on dataset A does not generalize well as seen from its F1-scores evaluated on dataset B averaging at 0.32, which is not better than chance. The model trained on dataset B generalizes better, with an average F1-score of 0.686 evaluated on dataset A. Both show accurate results on the testing sets from the datasets they were respectively trained on. Overall, this indicates that dataset B has higher quality and varied data than dataset A in this context.

The final model trained on both dataset A and B has very solid overall results when evaluated on either of their testing sets. The overall F1-score is high, set at 0.87. However, higher variety of hiking trail data would need to be gathered to evaluate how much this model truly generalizes. As showcased by the evaluation results for the two partial models, both datasets possess a certain level of bias between their training data and testing data.

Images are computed through a Raspberry Pi 3 (Model B) with 1GB RAM and a 1.2 GHz 64-bit quad core processor to demonstrate the model being computationally cheap enough to run on an embedded system; performing predictions with an average runtime of 514 milliseconds.

Various trail images are gathered and processed by the final model, to illustrate predictions being performed on hiking trail images outside of dataset A and B.

(a) Examples of sections of varied trails that are predicted correctly. (Street view data: Google 2020).

(b) Examples of sections of varied trails that are predicted incorrectly. Images with terrains completely unlike the training dataset's can lead to erroneous predictions. (Street view data: Google 2020).

Figure 4.2: Illustrations of predictions from the final model. The green arrow represents one of the three directions (left, right or straight) with the highest confidence. The red arrow represents an estimation of a direction angle generated from the mean of the confidence values.

5. Conclusion

This paper aimed to build a machine learning solution for hiking trail perception, while remaining viable to use for low resource platforms. The MobileNetV2 architecture was suitable for this goal as a light-weight neural network that can be used for image classification. The final trained model showed moderately successful testing results, with a mean F1-score of 0.87.

Future work would expand on model testing: such as building a dataset that labels directional angles for paths instead of just one of three direction, as to be able to test the accuracy of the "mean directional angle" predictions of our model. Furthermore, various state-of-the-art trail tracking algorithms (see Section 1) could be implemented to compare their accuracies and performances to this model.

A dataset of hiking trail images was built and uploaded, labelled as either "left", "right" or "straight", representing the trail's direction in relation to the image's perspective. One major limitation of the dataset is that it mostly contains very similar environments. Another dataset [12] was used to somewhat alleviate this limitation, but in order to generalize the model further, much greater variety of training data should be gathered from various environment types, such as mountains, deserts, shorelines, and snowy terrains.

Another limitation comes the lack of challenging trail footage provided by the dataset; most of the dataset contains trails that would be trivial for a human to navigate. While this leads to models trained on this data to be somewhat successful on simple trails, it could lead to inaccuracy on much more challenging trails. As many people tend to get lost on hiking trips yearly [4], the advantage of deep neural networks to outmatch humans should be more of a focus for future work, to be more useful for humans, and not just for simple robot navigation purposes. Data from more difficult to navigate trails would need to be gathered to evaluate the extent to which the model generalizes to them, and to use it as training data.

Further testing of the model could be performed through a field test on a different hiking trail, making predictions live on a low resource device. Assuming model generalization is improved sufficiently, a low-altitude drone could be made to follow a trail vastly different than those it has been trained on as a test.

6. Retrospective

When the project originally started, a simple convolutional neural network was going to be trained, with an emphasis on model accuracy. However, as the dataset was limited, creating a generalized model was untenable and a focus was instead put on improving performance for the model to be usable on low resource platforms, and demonstrating how adding data variety greatly improves model generalization.

The main disciplines required for this project were in computer vision and machine learning. I had little practical experience in both, so this served as the first application of the theoretical knowledge I had for a real project. Because of this lack of experience, a lot of time was allocated to designing the model in the Gantt Charts (see Appendix B). While the schedule did not need to be modified for most of the project, the final sections of the Gantt Chart for training and testing ultimately were changed because of time constraints; total training time was severely underestimated, and debugging failed training sessions ate away at the remaining time.

Building, managing, and uploading a dataset completely from scratch proved to be quite challenging, with many recording trips having to be redone because of minor issues such as wobbling cameras, and dirtied lenses leading to unsalvageable footage. Ultimately, limitations of the dataset led to including another dataset for the project [12].

A final field test on a new hiking trail was considered, where it was planned to record the best model performing predictions from a video feed, having it loaded on a low resource platform such as a Raspberry Pi 3 (Model B) with a camera module, but this ultimately had to be cancelled because of recent quarantining policy against COVID-19. At the start of the project, a risk that would lead to an unavailability of hiking trails was considered in the risk management table (see Appendix C) and so we were able to follow the measures we had planned, by performing all testing on data from home.