Learn 2 Rage: Experiencing The Emotional Roller Coaster That Is Reinforcement Learning

Lachlan Mares [*] , **Stefan Podgorski** and **Ian Reid**
Australian Institute of Machine Learning, University of Adelaide
[3]University of Adelaide
{lachlan.mares, stefan.podgorski, ian.reid}@adelaide.edu.au

Abstract

This work presents the experiments and solution outline for our teams winning submission in the Learn To Race Autonomous Racing Virtual Challenge 2022 hosted by [AIcrowd,]. The objective of the Learn-to-Race competition is to push the boundary of autonomous technology, with a focus on achieving the safety benefits of autonomous driving. In the description the competition is framed as a reinforcement learning (RL) challenge.

We focused our initial efforts on implementation of Soft Actor Critic (SAC) variants. Our goal was to learn non-trivial control of the race car exclusively from visual and geometric features, directly mapping pixels to control actions. We made suitable modifications to the default reward policy aiming to promote smooth steering and acceleration control.

The framework for the competition provided real time simulation, meaning a single episode (learning experience) is measured in minutes. Instead of pursuing parallelisation of episodes we opted to explore a more traditional approach in which the visual perception was processed (via learned operators) and fed into rule-based controllers.

Such a system, while not as academically "attractive" as a pixels-to-actions approach, results in a system that requires less training, is more explainable, generalises better and is easily tuned and ultimately out-performed all other agents in the competition by a large margin.

1 Introduction

As autonomous vehicle technology advances and becomes part of the new normal there is greater importance for autonomous vehicles to adhere to safety standards. Whether the settings is urban driving or high-speed racing similar principals apply. Racing demands that a vehicle operate on the edge of its physical limits at all times dramatically increasing operational risks. Safe operation becomes critical, especially when any slight infraction could lead to catastrophic failure. Given financial investments in racing vehicles is high, autonomous racing in a controlled simulated environment can serve as a proving ground for development of safe learning algorithms.

Figure 1: Learn 2 Race simulator

The Learn-to-Race competition was a simulator-based challenge in which the aim was to build an autonomous agent to control a virtual racing car around a track. Agents were given prior learning opportunities to drive around a specific track, and then limited exposure to a new, different track, for which the goal was to achieve the fastest time without incurring safety infractions.

This paper describes our solution to this problem which achieved first place in single camera, and fastest overall track time in the 2022 edition of the challenge. We discuss the key features of our solution, as well as providing details of an academically more appealing, but significantly less successful approach based more directly on reinforcement learning. We use the contrast in these approaches to infer some lessons about learning for driving and other complex control tasks.

1.1 Learn To Race competition

The Learn-to-Race competition provides contestants with a Gym-compliant framework that leverages a high-fidelity racing simulator developed by [Arrival,].

It comprised two rounds. In round one participants developed and evaluated their agents on Thruxton Circuit, which

[*]Contact Author

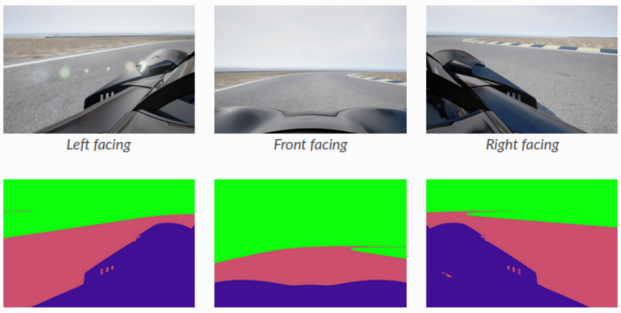


Figure 2: Observation images and segmentation masks

was included with the Learn-to-Race environment. Due to the local availability of the round one evaluation track it was possible to perfectly fit an agent given the unbounded amount of practice time.

In stage two agents were evaluated on an unseen track, the Vegas North Road Circuit. Evaluation consisted of a one hour practice period, then three laps of the circuit. The evaluation score considered safety infractions during training, together with safety infractions plus average speed for the additional three laps.

Action space

Agents execute actions within a simulated environment which models the dynamics of a vehicle and generates visual imagery from an agent-centric viewpoint. The action space comprises steering and acceleration control both with a continuous range of -1.0 to 1.0 representing full left to full right and maximum braking to maximum acceleration.

Observation space

The Learn-to-Race simulator provides competitors with multi-modal sensory inputs. The simulation environment provides for various observation data. During evaluation the only observation data available is the sequence of RGB images from a driver-centric perspective and the vehicle velocity. In addition, during development agents have access to semantic pixel labels (which designate where the road is in each image; see Figure 2), and the full vehicle state.

Racetracks

The simulation environment included two real-world tracks. The first was Thruxton Circuit used for round one evaluation, modeled from the track at the Thruxton Motorsport Centre in the United Kingdom. The second is Anglesey National Circuit, located in Ty Croes, Anglesey, Wales.

For round two, the evaluation competition servers hosted an unseen track Vegas North Road, located at Las Vegas Motor Speedway in the United States.

1.2 Our approach

The objective of the Learn-to-Race competition is to push the boundary of autonomous technology, with a focus on achieving the safety benefits of autonomous driving.

We focused our initial efforts on implementing RL baselines [Raffin *et al.*, 2021] such as Twin Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor Critic (SAC). In particular our goal was to learn non-trivial control of the race car exclusively from visual features, directly mapping pixels to control actions. We made suitable modifications to the default reward policy aiming to promote smooth steering and acceleration control. These experiments and results are outlined in Section 2.

Nevertheless, the framework for the competition provided only for real time feedback, meaning a single episode (learning experience) is measured in minutes. Unless the training could be done via massive parallelisation of episodes (which we did not seek to do) the amount of training resources required was a significant limitation.

As a baseline we decided also to explore a more traditional approach in which the visual percepts were processed (via learned operators) and fed into a rule-base controller. Such a system, while not as academically "attractive" as the pixels-to-actions approach, results in a system that requires less training, is more explainable, generalises better, is more easily tuned. Our system solution and experimental results are outlined in Section 3.

We conclude the paper with a discussion of the relative strengths and weaknesses of the two approaches, lessons learned, and likely directions for future development.

2 Approach 1: Reinforcement Learning

Our initial solution aims to use reinforcement learning (RL) to learn control commands based solely and directly from the RGB images "observed" by the agent. We term this approach a direct pixels-to-actions solution. While it is indeed possible to learn to race quickly and safely on a single track using this approach [Fuchs *et al.*, 2020], a significant weakness of naive application of this approach is over-fitting to aspects of the imagery that are specific to the training track, and therefore a failure to generalise to a new track. For this reason, and taking inspiration from [Loquercio *et al.*, 2021] we preprocess the images in a number of ways, computing optical flow, scene segmentation and/or edge detection, as a means to train the system with features that generalise across domains and weather/lighting conditions. We further compress these features using a Variational Auto-Encoder (VAE) by only using the latent vector and discarding the decoder at run-time.

During the experiments the Soft Actor-Critic (SAC) [Haarnoja *et al.*,] RL algorithm was used as the agent. A SAC agent comprises of two Multi-Layer Perceptrons (MLP), one is the actor and the other critic. The input to our SAC agents was obtained by passing the raw (multi-channel) image data through some pre-processing module/s to obtain a feature vector. The feature vector is a compressed representation containing relevant and important information about the environment.

In all our experiments the pre-processing module used was VAE. An Auto-Encoder (AE) is a deep-neural network designed to encode an image into a latent representation then reconstruct the image from the representation. A VAE was chosen in preference to an AE as the encoding distribution is regularised during training. We expect that this regularisation helps learn latent representations with better domain generalisation properties. We describe how these pre-processing

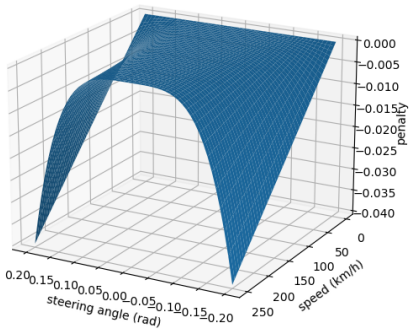


Figure 3: Steering penalty function for steering angle and speed.

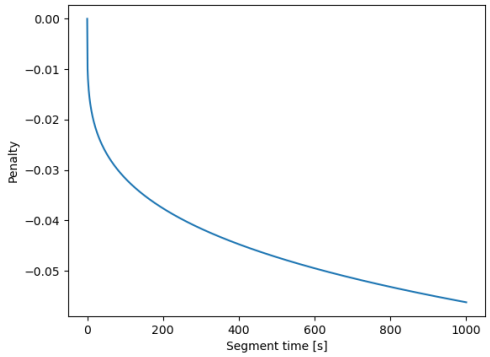


Figure 4: Segment penalty function for time agent is in a segment.

elements are learned and used in the following sections.

2.1 Continuous SAC

In this section we detail our experiments and trials using SAC with acceleration and steering action in continuous space using only VAE encoded RGB images for observations.

SAC with FLARE

To improve the sample efficiency when training SAC Flow of Latents for Reinforcement Learning (FLARE) [Shang *et al.*, 2021] was integrated by taking the difference between the current and last timesteps latent feature vectors. SAC with FLARE (SAC-F) was used in all of the following RL experiments.

SAC-F with Modified Reward Policy

The default reward function for the competition was an adaption of the reward function presented in [Fuchs *et al.*, 2020]. The reward function has two components, a) track progression calculated based on centreline and b) avoiding track boundaries. We found during training the agent heavily favoured the track centre. Observed behaviour was the vehicle oscillating left and right trying to maintain track centre.

Oscillating either side of track centre is inefficient, unstable and likely create unnecessary accidents and infractions during racing. With this in mind a modification to the reward was made to penalize the agent for making large steering changes. We formulate a steering penalty term:

$$\rho_{\text{steer}}(\alpha, s) := -\frac{s\alpha^4}{10}$$

where α is the agent's speed and s is the steering input, graphed in Figure 3. This penalty was added to the reward at each time step with the intention of encouraging the agent to find a balance between smooth steering control and staying on the centerline.

Learn-to-Race is a race, staying on track is necessary however the agent needs to drive as fast as possible. We found that using the default reward function did not result in fast track progression. To promote speed a penalty term was added to the reward function at each time step. This addition penalised the agent for the amount of time spent in each track segment.

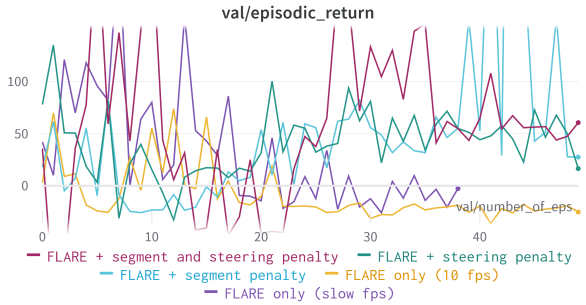


Figure 5: Cumulative reward for validation runs using the modifications to SAC.

To implement this penalty a timer $t \in \mathcal{R}$ was started at the beginning of each track segment as the agent entered it. The penalty accumulated defined by:

$$\rho_{\text{segment}}(t) := -\frac{t^{1/4}}{100}$$

while the agent was inside the segment with t resetting at the start of the next segment. The form of the penalty is shown in Figure 4.

Results

Overall the performance of SAC-F with additional penalty terms was poor giving no successful completion of the track without safety infractions for any of the modifications. The modifications to the reward functions showed promise for improving the validation episodic return. The small affect the additional penalty had on the cumulative reward shown in Figure 5 was seen after around 20 validations where the SAC-F agent trained with the penalties consistently returns positive reward.

2.2 VAE with Optical-Flow and Image Edges

The purpose of the VAE is to transform the input image feature space to a lower dimensional latent space. The latent space is a compressed version of the input without unnecessary information. The VAE used in the previous methods were training on a data set of RGB images that were collected

from the vehicles camera during manual control on the tracks that were available to entrants during the competition.

However, the final stage of the Learn-to-Race competition had entrants competing on an unseen track whose image distribution would not be the same as the training image distribution. The mismatch in the image distributions would cause poor generalisation due to over-fitting on the data set during training.

Therefore, the input images need to be pre-processed to extract features that describe the geometry of the observation for any track irrespective of specific image pixel values recorded by the camera. The new pre-processed images can then be used as the VAE input to produce abstract latent vectors for the agent.

The following methods of image pre-processing for abstract latent vectors were trialled: Binary segmentation masks (discussed in Section 3.1), RGB representations of optical flow and, edge images.

Specific network details for the modified VAE can be found in Appendix 5.1.

Optical Flow

Optical flow was used to add information regarding the agents movement and to add geometry. The optical flow was generated using the Farnebäck method [Farnebäck, 2003] and then converted into a three-channel RGB image.

Edge Images

The images edges were used to add more detailed geometry information about the track which would be transferable by stripping away the rendered textures and leaving only the simulators geometry behind. The image edges were produced using two methods: Canny edge detection [Canny, 1986] and edge detection using the Söbel-Feldman operator.

Custom VAE with Combined Inputs

The standard VAE used as a baseline in the Learn-to-Race competition needed to be modified and retrained to fit the new distributions formed by the abstract observations and their combinations. The baseline VAE was modified by increasing the depth of the network with an additional convolutional block at each layer, adding residual connections to the encoder and decoder convolutional blocks, using Mish activations [Misra, 2019] instead of ReLU, adding batch normalization to each convolutional block and initialising the VAE's weights using Xavier initialisation [Glorot and Bengio, 2010]. No residual connections bypassed the bottleneck. The bottleneck connection required its output to be normalized using batch normalization for stability during and training.

The modified abstract VAEs were trained on a NVIDIA 2080ti with cosine annealing [Loshchilov and Hutter, 2016] for 1000 epochs starting with an initial learning rate of 0.001. A number of variations of input abstraction channels were trained which were combined to add information for the latent space. The variations we tested were Canny edges only, then gray scale image, see Figure 6, each combined with the RGB optical flow.

SAC-F Using Abstracted Latents

Three methods were chosen to trial, the baseline RGB VAE with FLARE [Buslaev *et al.*, 2020], RGB optical flow over the previous and current frame with an additional Söbel image edge channel, and Canny edges only. The combined segmentation masks and RGB optical flow did not appear to reconstruct a detailed enough image and was not trialled which indicate that the latent vectors did not contain enough information. The methods were trialled for 10-20 hours on a NVIDIA 2080ti with a frame rate of 5 fps. Overall, the use of the abstracted latents marginally improved the total distance travelled (Figure 7 (a)) and episodic return (Figure 7 (b)) during validation on Thruxton when compared to the baseline method. However, the average speed (Figure 7(c)) was negatively affected by the addition of the abstract latents.

2.3 RL approach summary

Due to the general applicability of model free RL algorithms should be the ideal candidate for complex control tasks such as autonomous racing applications. The ability to learn from experience, adapt and find unique policies in theory should be able to out-perform any rule based solution.

The issue we faced during the competition is the optimal policy is learnt from the environment through extensive exploration, this process however takes thousands of experiences from thousands of environment steps costing time and computing resources that we were unwilling to commit given the one hour time constraint in round two.

3 Approach 2: Traditional methods

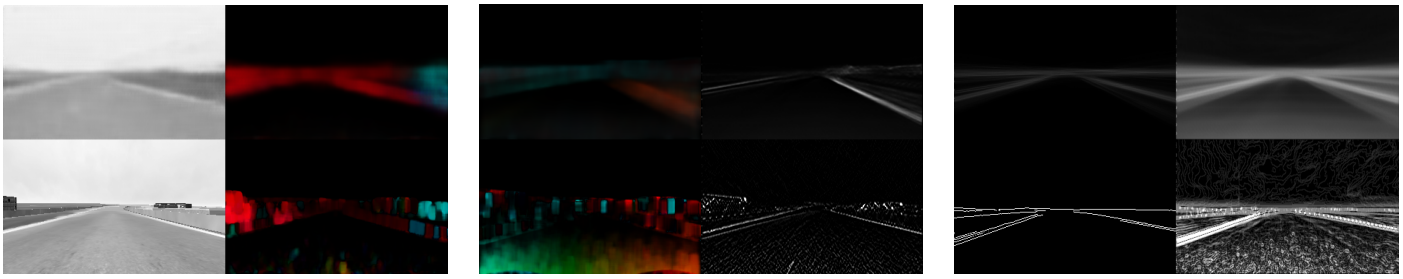
As a minimum baseline we also implemented a more traditional approach to autonomous agent control. This approach uses a combination of (learned) image processing to achieve situation awareness (track location, position, etc) and rule-based control.

In particular, we implemented two localisation modules: the first identified where the centreline of the road in current observation, as the primary sensor signal to influence steering angle. The second module identified which zone of the track the agent was currently traversing. This was used to inform the agent of upcoming acceleration and braking zones. Such zones can only be identified in the broader context of a memory of the whole track, since they often cannot be disambiguated using the immediate visual stimulus alone. Without positional awareness the acceleration controllers ability to safely regulate speed was constrained by the camera field of view.

Given the simulator provided segmentation masks a logical choice was to use a semantic segmentation network for track surface identification. The segmentation network accurately classified image pixels as either track or not-track. With observable pixels classified, key geometric properties of the observation could be calculated. The properties of the observation can then be passed to steering and acceleration controllers.

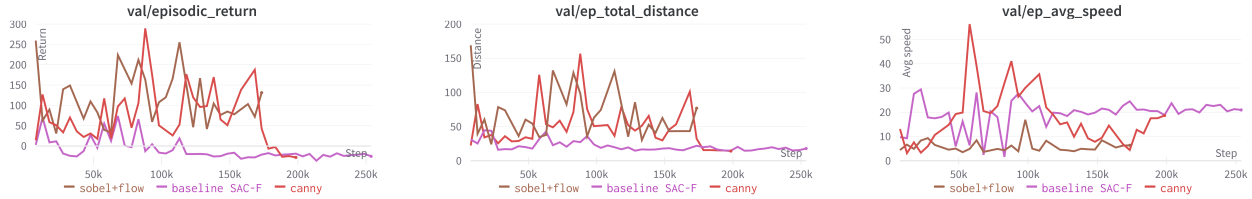
3.1 Semantic segmentation

This section discusses our semantic segmentation model used to extract the drivable area ahead of the agent. We first dis-



(a) (b) (c)

Figure 6: Reconstruction results: Top row shows the reconstruction and bottom row is the ground truth. (a) gray scale and RGB flow (b) Söbel edge images and RGB flow (c) Söbel edge images and Canny edge images



(a) (b) (c)

Figure 7: Episodic return for the each of the trialled VAEs. (a) Episodic return is generally higher for the abstract latents than the baseline method. (b) total distance travelled in the validation episode was in general higher than the baseline method and (c) average speed was in general higher for the baseline method than the abstract latents.

cuss the model architecture utilised for agent perception then discuss the offline training regime. Finally we present the fine tuning used by our agent during the one hour training period before final evaluation.

Segmentation model architecture

The semantic segmentation model required a balance between low inference time and high accuracy. The architecture chosen was a custom PyTorch [Paszke *et al.*, 2019] implementation consisting of an EfficientNet-V2-Small encoder paired with a Feature Pyramid Network (FPN) decoder.

EfficientNetV2 is a convolutional neural network that has been optimised to increase training speed and maximise parameter efficiency. To develop these models, the authors [Tan and Le,] use a combination of training-aware neural architecture search and scaling, to jointly optimize training speed. EfficientNetV2 utilises a new Fused-MBConv in conjunction with the MBConv present in EfficientNet. The EfficientNetV2 encoder used for the challenge was adapted from an existing PyTorch repository [Han, 2022].

FPN is a feature extractor for segmentation networks proposed by [Lin *et al.*, 2016]. The FPN feature extractor generates multiple feature map layers (multi-scale feature maps) with higher quality information than the regular feature pyramid for object detection.

Specific network details can be found in Appendix 5.2.

Segmentation model training

The segmentation model was trained using 384x512 greyscale images to reduce the expected domain gap between unobserved tracks. Extensive image augmentations were used

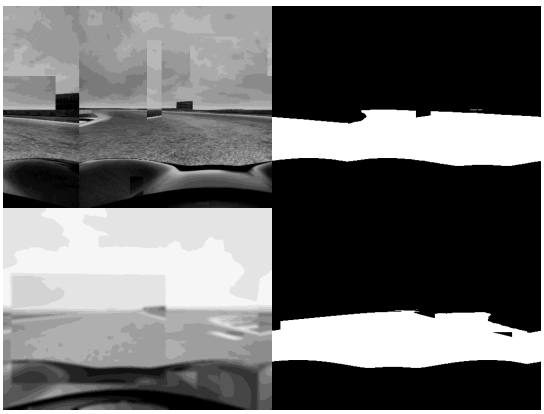


Figure 8: Augmentation Examples

to further extend the captured 25000+ training examples. Image and mask augmentation was primarily handled with the Albumentations library [Buslaev *et al.*, 2020]. Augmentations included Horizontal-Flip, Blur, CLAHE, Posterize, Random Contrast and Random Brightness. Additional custom augmentations included axis roll and random cropping. See Figure 8 for some training samples.

The segmentation model was trained using a standard Adam optimiser and a custom implementation of Dice Loss:

$$\text{Dice Loss} = 1 - \frac{2 * TP}{2 * TP + FP + FN}$$

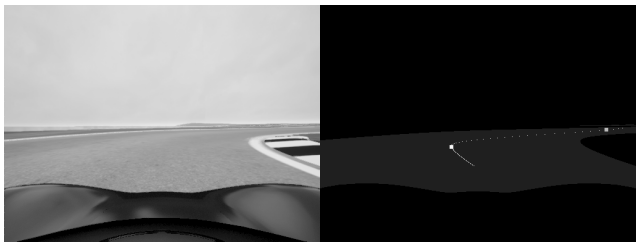


Figure 9: Upper/Lower bounds of usable centreline pixels

Segmentation model training during evaluation

The Learn To Race Challenge allowed participants to see the test track for one hour prior to evaluation. We utilise this time to fine tune the segmentation model to ensure high accuracy predictions. During this training period our agent conservatively drives around the track inferring the drivable area. The loss between the inferred prediction and the provided ground truth is then calculated. If the loss is higher than a tunable threshold it is saved to be used for fine tuning later in the evaluation period. It was found that to ensure stability when training images and masks from Thruxton and Anglesey also needed to be included in the fine tune training.

3.2 Steering controller

The steering and acceleration controllers both rely on predictions from the segmentation model. An estimate of the track centreline is calculated using the boundaries of the predicted road surface shown in Figure 9. The steering angle was calculated in pixel coordinates rather than transforming to Cartesian coordinates. Based on the vehicles current velocity a selection of centreline pixels were used to calculate the relative angle RA for the next environment step:

$$RA = \frac{\arctan\left(P_j, (P_i - (w/2))\right) - \pi/2}{\pi/2}$$

where P_{ij} is the centreline pixel indices, and w is the image width.

The relative steering angle was fed into a Proportional Integral Differential (PID) controller.

3.3 Acceleration controller

To approximate future time-steps the track centreline pixels were split into discrete chunks. The chunk size was chosen so that it represented expected future vehicle positions in the observed image. The chunks were then used to compute an estimation of the track curvature at each future step. A Gaussian weighting parameter W was used to weight the future track curvature and current velocity:

$$\mathrm{W} = 1 - \frac{1}{c}\sum_{j=1}^{c}\left(\exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)\right)$$

where c is the number of chunks, μ is c evenly distributed values between 0 and 1, x is the current velocity as ratio of maximum velocity set-point, and σ is the the spread of individual chunks.

Track	fps	local kph	fps	eval kph
Vegas North Road			6.4	70.853
Anglesey National	7.5	85.89		
Thruxton	7.5	132.56	6.4	126.350

Table 1: Track times without location classifier, fps frames per second, local/eval (AIcrowd) refers to simulator location

Layer	Input Size	Output Size	Activation
Input	feature_size	feature_size	Tanh
Hidden 1	feature_size	512	ReLU
Hidden 2	512	256	ReLU
Output	256	classes	Softmax

Table 2: MLP Layers used for location classification

Taking inspiration from human drivers the acceleration controller used a discrete set of actions. The simplification of the action space permitted the use of a simple Bang-Bang controller to output one of five states given the current velocity and W. Action space used values [-1, 0, 0.2, 0.4, 1], 0.2 and 0.4 were used if current steering angle exceeded 0.5 and 0.3 respectively.

3.4 Results without track localisation

Work on track localisation started towards the end of round one of the Learn-to-Race competition. Round one submission could have certainly been improved with localisation, but given finishing in the top 10 granted access to Round two it was not pursued. The results for the three tracks pre-localisation are shown in Table 1

3.5 Track localisation with classification models

Without localisation the vehicles speed is limited by the track in which it can observe. Removing the threat of a tight corner appearing unexpectedly while traveling at top speed allows the agent to travel safely while also obtaining higher average speeds.

Experiments were conducted with three different implementations of location classification, all using a neural network. The first utilises the track segments provided alongside the simulator, the second uses an arbitrary number of segments spread out around the map, and the third requires hand labeling of distinct zones.

Classification model architecture

The classification model used during experimentation was a two layer MLP. Input to the classification model was a feature vector extracted from the semantic segmentation model. The architecture, dimensions and activation functions are listed in Table 2.

Specific network details can be found in Appendix 5.3.

Experiment 1: Environment track segment

The goal of this experiment was to train a classification model to recognise the current track segment provided by the simulator. The provided code base which runs experiments tracks the current segment. Probing this code at run time allowed us to label samples with the current segment. The model

was trained using 5000 samples spanning the 10 segments of Thruxton Circuit. The model achieved an accuracy of 99.76% on the 5000 samples collected. The errors observed during evaluation occurred at the boundary where two track segments met.

This classifier was not utilised in round two as knowing the vehicles location at such a coarse scale offered no benefit predicting upcoming difficult track sections.

Experiment 2: N Sections

The goal of this experiment was to train a classification model to recognise up to N discrete track sections. Track sections were defined by evenly selecting key-points along the tracks centreline, a JSON file containing centreline points (x, y) was included with the simulator. The N key-points (key_x, key_y) were then used to divide training samples into sections:

$$\text{section} = \arg\min \sqrt{(\text{key}_x - x)^2 + (\text{key}_y - y)^2}$$

The model was trained using 10000 samples spanning 40 sections. The model achieved an accuracy of 96.4% on samples gathered from Thruxton circuit. The observed errors during evaluation were not confined to the boundaries where two track sections met as with Experiment 1. The observation errors were able to be corrected with a transition filter which constrained section changes to occur in sequence.

This approach was a viable option but was not utilised in round two due to a preference for the method outlined in Experiment 3.

Experiment 3: Specific zone

The goal of this experiment was to train a classification model to recognise specific track zones deemed difficult or required actions beyond recognition of sensors. Track zones were categorised as one of three discrete classes:

- 0, High speed areas, straights and exits from corners
- 1, Approaching moderate corners at mid velocity, tight corners at low velocity
- 2, Approaching moderate and tight corners such as chicanes and hairpins at high velocity

Selecting track zones was a manual process based on intuition and experiments. For example a long straight followed by a chicane would is labelled as zone 2 as the agent will be approaching at high speed and need to slow down. The defined zones were specified as boxes $(xmin, ymin, xmax, ymax)$ and labeled [0, 1, 2]. Setting zones for each track is an iterative process starting conservative then altering them based upon feedback from observation, evaluation logs and results. Experimentation was conducted on Thruxton and Anglesey National Circuits, zones are shown in Figure 10.

The model was trained using 10000 combined samples, 5000 from both Thruxton and Anglesey National Circuits. The model achieved an accuracy of 99.62% during evaluation laps of Thruxton, and 99.69% during evaluation laps of Anglesey National. As seen in Experiment 1 the observed errors during evaluation occurred where two track zones met. Out of the three experiments the specific zone classifier was

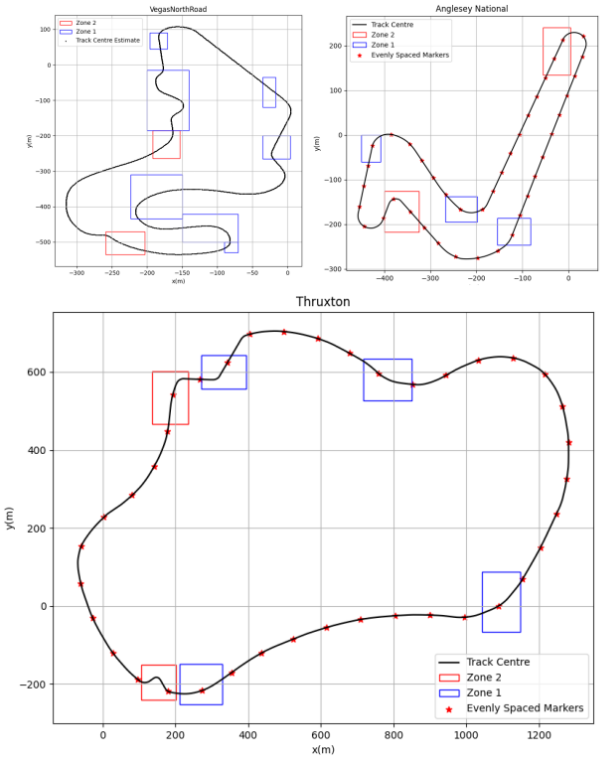


Figure 10: Zone boundaries for all three tracks

the easiest to configure, had the best generalisation and was trainable within the one hour training window.

This version of location classifier was utilised in the round two of Learn to Race, and was a major contributor to winning the competition.

Classifier model training during evaluation

To collect the samples required to train the localisation classifier the vehicle was set to follow the estimated centreline of the track at up to 10m/s. At each environment step the current zone was calculated using the zone boundaries shown in Figure 10, then the feature vector from the already fine tuned segmentation model and zone were saved. A total of 4500 training samples were collected. Since the majority of the collected samples are within zone 0 balancing of classes was required. We collect an additional 500 samples of zone 1 and 2 to combat this.

No augmentations were applied to the feature vector during training and no train-validation split was used. The intention during training was to over-fit the model to the data. Adam optimiser and cross entropy loss were used for training as per aforementioned experiments. The model accuracy recorded for our best submission was 97.7% after 150 epochs, the lower than expected accuracy led to slightly inconsistent zone predictions. Without local verification it is unclear how much effect the mis-classifications had on track times.

Results with track localisation

Of the three classifiers Specific Zone was chosen for round 2. The work implementing the localisation proved to be crucial

Track	fps	local kph	fps	eval kph
Vegas North Road			5.1	92.54
Anglesey National	7.0	103.97		
Thruxton	7.0	147.12		

Table 3: Track times with location classifier, fps frames per second, local or evaluation (AIcrowd) refers to simulator location

for posting the fastest lap time in round two and ultimately winning the Learn-to-Race competition. This implementation posted a 21.6kph improvement in average speed compared to initial submission. The results for the three tracks with localisation are shown in Table 3.

4 Conclusion

In this paper, we have presented the experiments and final approach taken in the Learn-to-Race challenge for safe autonomous driving 2022.

In summary, the RL agents from our experiments were inefficient to train and offered no significant improvement compared to the provided Learn-to-Race baseline agents. The inability to effectively explore and sample the unobserved evaluation environments resulted in our agents exhibiting undesired behaviours that were prone to safety infractions.

The major benefit of our traditional approach was its ability to generalise, requiring minimal training time to perform well in unobserved environments. There are still a few key areas that can be explored to increase performance for future work these include a) streamlining of code to increase the observation and controller update rate, b) implement more sophisticated steering and acceleration controllers, and c) explore bootstrapping an RL agent with current approach essentially using RL as fine-tuning mechanism.

References

[AIcrowd,] AIcrowd. Crowdsourcing ai to solve real-world problems. https://www.aicrowd.com/.

[Arrival,] Arrival. Explore arrival products and technologies. https://arrival.com/world/en.

[Buslaev *et al.*, 2020] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[Canny, 1986] John F. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[Farnebäck, 2003] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Fuchs *et al.*, 2020] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürr. Superhuman performance in gran turismo sport using deep reinforcement learning. *CoRR*, abs/2008.07971, 2020.

[Glorot and Bengio, 2010] Xavier Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, 01 2010.

[Haarnoja *et al.*,] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. In *ICML*.

[Han, 2022] Han. Efficientnetv2-pytorch. https://github.com/hankyul2/EfficientNetV2-pytorch, 2022.

[Lin *et al.*, 2016] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2016.

[Loquercio *et al.*, 2021] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021.

[Loshchilov and Hutter, 2016] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2016.

[Misra, 2019] Diganta Misra. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, 2019.

[Paszke *et al.*, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[Raffin *et al.*, 2021] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[Shang *et al.*, 2021] Wenling Shang, Xiaofei Wang, Aravind Srinivas, Aravind Rajeswaran, Yang Gao, Pieter Abbeel, and Misha Laskin. Reinforcement learning with latent flow. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 22171–22183. Curran Associates, Inc., 2021.

[Tan and Le,] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training.

5 Appendix

This appendix adds further implementation details to the networks used during the competition.

5.1 Variational Auto-Encoder Architecture

The baseline VAE provided in the competition was modified for the optical flow and edge images as described in section 2.2. The convolutional blocks are summarised in Table 4, 5 and a diagram detailing the layers of the blocks are shown in Figure 11.

During training on the combined abstract input channels the network tended to become unstable which prevented a useful output. The fully connected layers were modified with normalisation to overcome the instability. The modification is shown in Figure 12.

Stage	Block	Channels	Kernel	Stride	Pad
0	Conv_{00}	4, 64	3	2	1
0	Conv_{01}	64, 64	3	-	same
1	Conv_{10}	64, 128	3	2	1
1	Conv_{11}	128, 128	3	-	same
2	Conv_{20}	128, 256	3	2	1
2	Conv_{21}	256, 256	3	-	same
3	Conv_{30}	256, 512	3	2	1
3	Conv_{31}	512, 512	3	-	same

Table 4: Modified VAE Encoder Architecture

Stage	Block	Channels	Kernel	Stride	Pad
3	Conv_{31}	in_ch, 512	3x3	2	1, 0
3	Conv_{30}	512, 512	3	3	1
2	Conv_{21}	512, 256	4	2	0, 1
2	Conv_{20}	256, 256	3	1	1
1	Conv_{11}	256, 128	3	2	2
1	Conv_{10}	128, 128	3	2	2
0	Conv_{01}	128, 64	3	2	2
0	Conv_{00}	64, 4	3	1	1

Table 5: Modified VAE Decoder Architecture

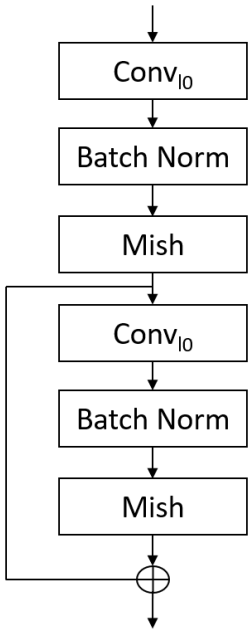


Figure 11: Detail of the modified convolutional blocks used.

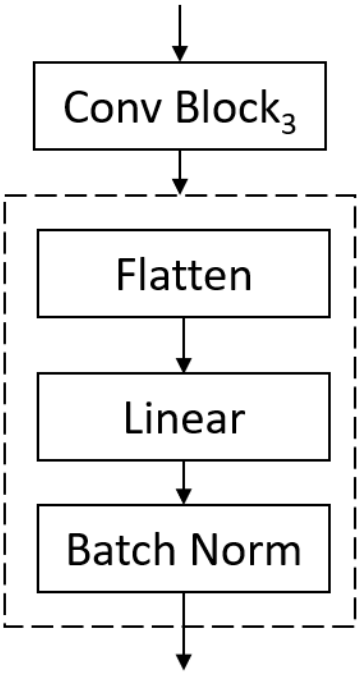


Figure 12: Detail of the modified fully connected blocks prior to re-parametrisation.

5.2 Segmentation Network Architectures

As mentioned in section 3.1 the semantic segmentation model used was a EfficientNetV2 small encoder, paired with a FPN decoder. The EfficientNetV2 blocks are summarised in Table: 6.

Stage	Block	Stride	Filters	Layers
0	Conv	2	24	1
1	Fused-MBConv1	1	24	2
2	Fused-MBConv4	2	48	4
3	Fused-MBConv4	2	64	4
4	MBConv4-SE0.25	2	128	6
5	MBConv6-SE0.25	1	160	9
6	MBConv6-SE0.25	2	256	15

Table 6: EfficientNetV2-S Encoder Architecture Detail

The difference between MBConv and Fused-MBConv are shown in Figure 13.

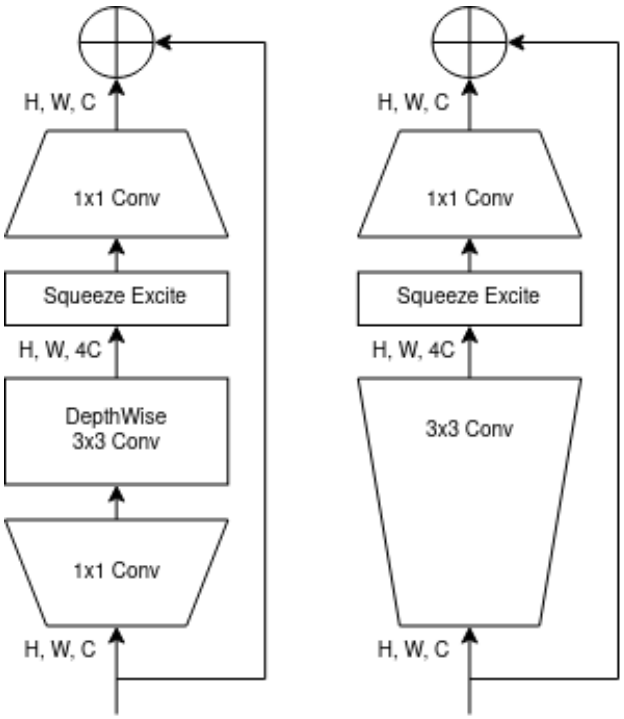


Figure 13: MBConv and Fused-MBConv

The FPN decoder uses convolutional filters passed from the encoder at different resolutions. Filters summarised in Table: 7, input image size was (384, 512, 1).

Stage from table 6	Resolution	Filters	Shape (h, w)
2	1/4	48	96x128
3	1/8	64	48x64
5	1/16	160	24x32
6	1/32	256	12x16

Table 7: Encoders convolutional filters passed to FPN

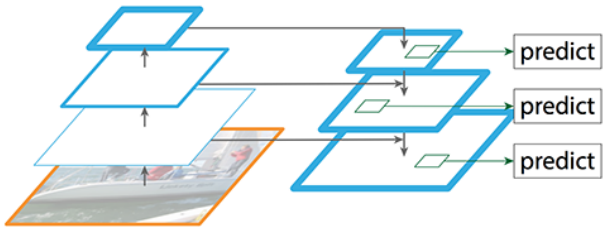


Figure 14: Feature Pyramid Network [Lin *et al.*, 2016]

5.3 Classification Network Architecture Detail

As mentioned in section 3.5 the classification model was an MLP. The input feature vector of length 49152 was created by reshaping the lowest resolution convolutional filters from the EfficientNetV2 encoder, see last row in table 7. The model specifics used for our round 2 submission is detailed in table 8.

Layer	Input Size	Output Size	Activation
Input	49152	49152	Tanh
Hidden 1	49152	512	ReLU
Hidden 2	512	256	ReLU
Output	256	3	Softmax

Table 8: MLP Layers used for location classification