# Autonomous Driving Agent in different environmental conditions

Rohan Asthana
July 18, 2021

**Abstract**

This project focuses on an objective to train and implement a deep learning agent that is able to drive in environmental conditions different on which it was trained by using two types of neural networks, i.e. Generative Adversarial Networks (GANs) and Convolutional Neural Networks (CNNs). We test different conditions such as snow, rain, night, rainy night, and snowy night for our analysis and conclude that the rainy night setting had the highest performance with a maximum achieved in-game reward as 747, whereas snowy day had the lowest performance with a maximum achieved in-game reward as 465.

## 1 Introduction

During the development of Autonomous driving agents, it is rather crucial to make the agent robust to different environmental conditions due to variance and uncertainty in real-life settings. Our project aims to solve this task in a simple virtual environment. With the state-of-the-art conditional GANs and CNNs, our project is able to develop such an agent. Our methodology can be divided into different steps. Firstly, we create different environments by changing the existing one. Secondly, we train the pix2pix model on images of changed environmental conditions (size 96x96x3), which outputs a corresponding normal environment image. Simultaneously, we train a CNN model, which when given a normal environment input frame image (grayscale image of size 96x96), outputs a possible key combination. After we have trained all the models, we use the models to predict the possible key combination from a test frame changed environment image.

## 2 Implemented Approaches

### 2.1 Changing the environment

We worked on five different changed environments apart from the normal environment- Snow, Rain, Night, Snowy Night, Rainy Night. To create all the environments, we changed the in-game attributes such as road colour and grass colour. We used the pyglet draw function to draw rain and snow in semi-random positions in the environment for rain and snow animations. We used diagonal long blue lines for rain and diagonal white short lines for snow to make it more accurate. We can see an example image from all the environments in Figure 1.

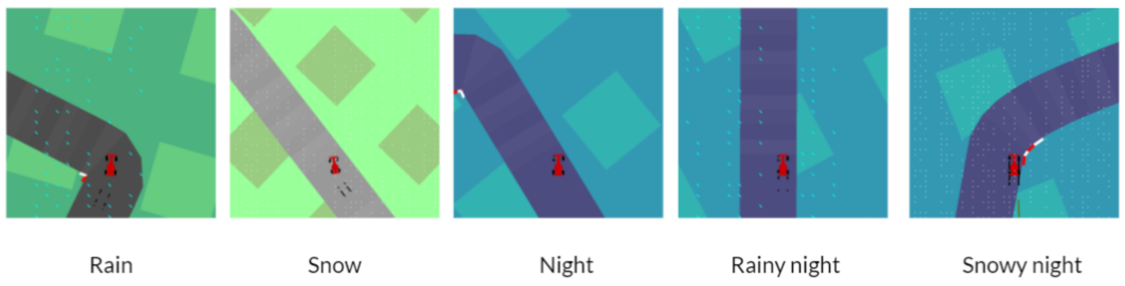


Figure 1: Different Changed Environments

### 2.2 Converting Changed Environment to Normal Environment

#### 2.2.1 GANs and pix2pix

GAN is a special kind of neural network consisting of two models, namely the Generator and a Discriminator. These types of networks were first introduced by Ian Goodfellow and other researchers in 2014. In these types of networks, the generator model is trained to generate new images (in our case, normal environment), and the discriminator model tries to classify images as either real (real normal environment) or fake (generated normal environment). [1] The training procedure for the generator is to maximize the probability of the discriminator making a mistake. The GAN framework corresponds to a minimax two-player game [2]

Pix2Pix is a special kind of GAN that learns the mapping from an image to another image and learns a loss function to train this mapping. [3] These networks are trained in such a way that they can successfully do an image to image translation. In the pix2pix model, the generation of the output image is conditional on an input. [4] The applications of these types of networks are converting Black and White image to RGB, converting the outline of an image to an actual image.
In our task, we use pix2pix for converting the changed environment image to the normal environment image on which our CNN model is trained.

#### 2.2.2 Adapting pix2pix

Pix2pix models are generally very deep and can easily overfit if provided by small-size images. In our case, as the images are only of resolution 96x96, therefore we need to adapt pix2pix to our task. We modified the Generator model by making it less dense and incorporating three encoder-decoder blocks instead of 7. We set the maximum filter size to 256. In Figure 2, we can see the general pix2pix pipeline. The discriminator has six convolution blocks, with each block containing a Convolution 2D layer, a Batch Normalization layer, and a LeakyRelu Activation layer.

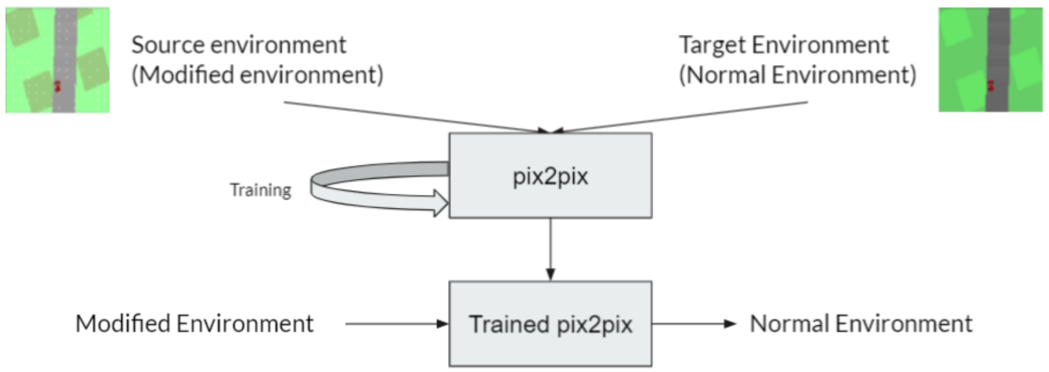


Figure 2: Overall Pix2Pix pipeline

#### 2.2.3 Creating training data for pix2pix

The pix2pix model requires a multiple set of two images- one changed environment image and one corresponding normal environment image of the same instance. It is vital for them to be the images of the same instance so that our pix2pix model can learn the mapping between the changed environment and the normal environment regardless of the road orientation and position of the car. We extract 6000 samples of images from the openAI environment while manually driving. After we have the normal environment images, we perform post-processing and change the extracted normal environment image to the target environment image (changed environment) by changing colours and adding animations. Now that we have multiple sets of two images, we can feed this data to our pix2pix model and train it.

#### 2.2.4 Training pix2pix

For training, we used our adapted pix2pix model and fed it with the normal environment, and changed environment images. We trained the model for 50 epochs for all environments except the snowy day, for which we trained it for 100 epochs because of the inability of the snowy day model to generalize. The loss function used was binary cross-entropy for the combined GAN model.

### 2.3 CNN Agent for normal conditions

#### 2.3.1 Convolutional Neural Networks

Convolutional Neural Networks are a kind of neural networks which are suitable to analyze images and visual content. A typical convolutional neural network comprises several convolutional blocks, with each block containing a convolutional layer, a pooling layer, and a batch normalization layer. After passing through a convolutional layer, the image becomes abstracted to a feature map. [5] When an image passes onto the Convolutional Neural Nets, the first few layers help extract the basic features of an image such as lines, edges, curves, but the deeper layers help extract more complex features. Therefore, it is helpful to use a deeper CNN if the image is of high resolution and contains complex features. However, in our case, as the image is relatively low resolution (96x96) and does not have complex features, we use a shallow CNN for our task. Using a deep CNN for our task might result in overfitting, which is undesirable.

#### 2.3.2 Creating training data for CNN

Since our task is to predict a possible key combination from a frame image, we need to preprocess the input before being fed into the CNN. For preprocessing the input, we downsampled the image to 96x96 to speed up the training process. We then converted the image from RGB to grayscale to reduce the size of the image even further. For each frame, a 96x96 NumPy array is appended to another array. We can see a sample preprocessed image in figure 3.
As the output of the CNN should be a possible key combination, we incorporated a function that records the key pressed for the corresponding frame and stores them in a one-hot-encoded array. For each frame, a 1x4 NumPy array was appended to another NumPy array.
We then manually drove the car in the normal environment and collected 100,000 samples of training data. The final input array was of size (100000x96x96), and the final output array (label array) was of size (100000x4). We then split the entire data into three separate parts- Training set, validation set, and test set. The training set included 60000 samples, whereas the validation set and the test set included 20000 samples. Now that we have the training data, our CNN agent is ready to be trained.

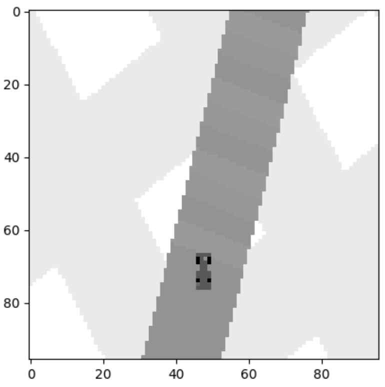


Figure 3: Preprocessed image

#### 2.3.3 Model architecture and training

For our task, a shallow CNN architecture is used. The architecture included three convolutional blocks, each with one convolutional layer, one max-pooling layer, and a Batch Normalization layer. In our task, multiple labels/keys can be the output. Therefore, the classes are mutually exclusive, and we use the sigmoid activation function instead of the softmax activation function. The sigmoid activation function allows the output neural nodes to be mutually independent of one another. That means that every output node will output the probability of it being true irrespective of the outputs of other nodes. Therefore, the sum of the output probabilities must not necessarily be 1. This method allows us to incorporate the "no-action" situation where the output is a zero array (i.e. no keys pressed) along with the situations where more than one keys have to be pressed. The loss function used was binary-cross entropy because we had to perform an independent binary classification of whether each node will be active or not. We trained the model for 200 iterations/epochs, and the model had a validation accuracy of around 70 percent.

#### 2.3.4 Further improvements

During the real-time testing of our agent, the car usually stopped around the corners and got stuck. The reason behind this behaviour was that when the car brakes too hard and stops, the CNN gets the same input indefinitely, which causes it to produce the same output indefinitely. We overcome this problem by introducing a hardcoded condition that restricts the agent to stop. We achieved that by providing it with a rule that if the speed of the car gets below 10, it should accelerate with the direction given by the network. This method solved the stopping problem and significantly improved the performance of the model.

## 2.4 Putting all the components together

Combining all the components, we first convert the normal environment to the changed environment during rendering by using the pyglet draw function. Secondly, we create training data for our adapted pix2pix model by post-processing the images extracted from manually driving in the normal environment. Then, we train the pix2pix model on 6000 pairs of normal and changed environmental conditions. Lastly, we train our CNN model on 60000 samples of training data.

## 2.5 Prediction and testing

Now that we have all the trained models, we can evaluate the agent in real-time. The prediction works as follows. Firstly, the current frame is extracted from our changed environment and is fed into the trained pix2pix, which outputs a normal environment image. This normal environment image is then fed to a trained CNN agent, which outputs the possible key combinations. Then, we use the 'step' function provided by openAI to move the car based on the key output. The overall pipeline can be seen in Figure 4.

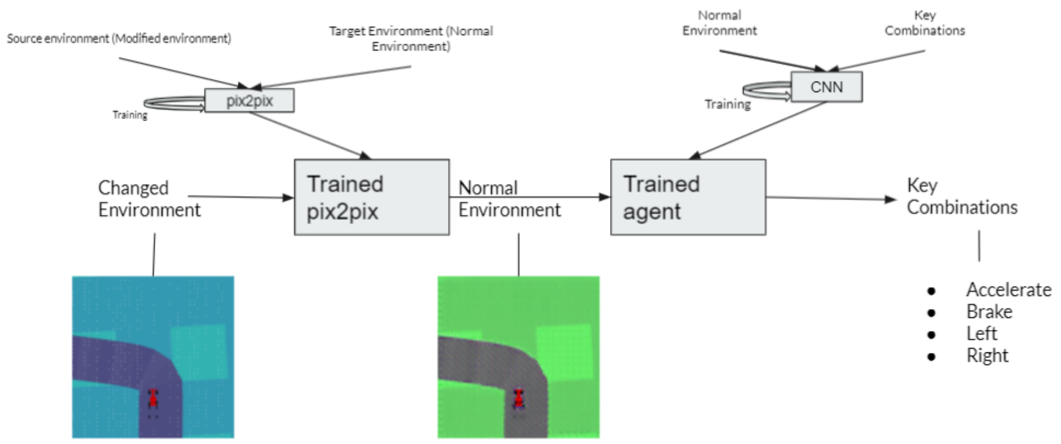
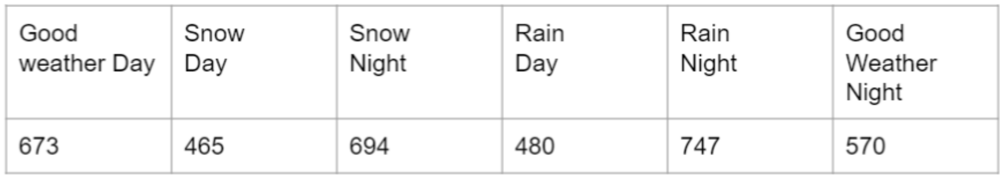


Figure 4: Overall Pipeline

## 3 Results

We tested a total of 6 environments, including the normal environment and the changed environments. For evaluating our agent, we used the maximum in-game reward achieved as a criterium for performance. The snowy day model performed the worst with the maximum in-game achieved reward of 465 whereas the rainy night model performed the best with the maximum in-game achieved a reward of 747. Performances of all the environments can be visualized in Figure 5.



| Good weather Day | Snow Day | Snow Night | Rain Day | Rain Night | Good Weather Night |
|---|---|---|---|---|---|
| 673 | 465 | 694 | 480 | 747 | 570 |

Figure 5: Results

## 4 Conclusion

In conclusion, we successfully were able to complete our objective to implement an autonomous driving agent able to drive in conditions on which it was not trained. The snowy day model performed the worst while the rainy night model performed the best. One possible reason for the snowy day model's lousy performance is that during the downsampling of the frame image, the snow became less dense and was hard to detect for the pix2pix model. As a result, the pix2pix model could not perfectly convert the snowy day image to the normal environment image, which, when fed to the CNN, resulted in outputting an incorrect key combination.

## 5 Future Directions

This work can further be improved by using various approaches. Firstly, using high-resolution images and deeper networks will help increase the performance as the deep learning models will detect the most delicate details in the frame. Secondly, this work can be further extended by incorporating more environmental conditions such as fog, thunderstorms. Lastly, better tuning of hyperparameters can lead to better performance of the agent.

## References

[1] J. Brownlee (2019) "A Gentle Introduction to Generative Adversarial Networks (GANs)" https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

[2] Ian J. Goodfellow, J. Pouget-Abadie, M. Mirza, B.Xu, D. Warde-Farley, S. Ozair , A.Courville, Y. Bengio (2014) "Generative Adversarial Nets" https://arxiv.org/pdf/1406.2661.pdf

[3] Philip Isola, J. Zhu, T. Zhou, A.A. Efros. (2018) "Image-to-Image Translation with Conditional Adversarial Networks" https://arxiv.org/pdf/1611.07004.pdf

[4] J. Brownlee (2019) "How to Develop a Pix2Pix GAN for Image-to-Image Translation. Machine Learning Mastery" https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

[5] W. Zhang (1988) "Shift-invariant pattern recognition neural network and its optical architecture". Proceedings of Annual Conference of the Japan Society of Applied Physics.