Controle de Um Veículo Autonomo em Ambiente Simulado CarRacing.v0

Alison de Oliveira Tristão
Otacilio Ribeiro da Silva Netto

4 de outubro de 2024

1 Introdução

Este trabalho tem como objetivo desenvolver uma rede neural para controlar um veículo em um ambiente simulado, utilizando dados gerados por um controlador automatizado. A proposta é aplicar técnicas de aprendizado de máquina para que o sistema aprenda a tomar decisões de direção de forma autônoma. O treinamento será realizado com imagens capturadas do simulador e entradas de teclado, ajustando os parâmetros de um modelo de rede neural convolucional. A avaliação do desempenho focará na evolução da acurácia durante as fases de treinamento e validação, com o intuito de verificar a capacidade do modelo de generalizar o controle do veículo em diversas pistas.

2 Formulação do Problema

Nosso objetivo é desenvolver uma rede neural capaz de controlar o carro no ambiente **CarRacing-v0**, da plataforma OpenAI Gym. Este ambiente foi projetado para testar algoritmos de inteligência artificial, particularmente em cenários que envolvem controle contínuo e visão computacional.

O jogo oferece uma visão 2D do circuito, a partir de uma perspectiva aérea, onde o agente (carro) deve percorrer a pista o mais rápido possível, evitando sair dos limites. O carro conta com três controles principais: **direção**, **aceleração** e **freio**.

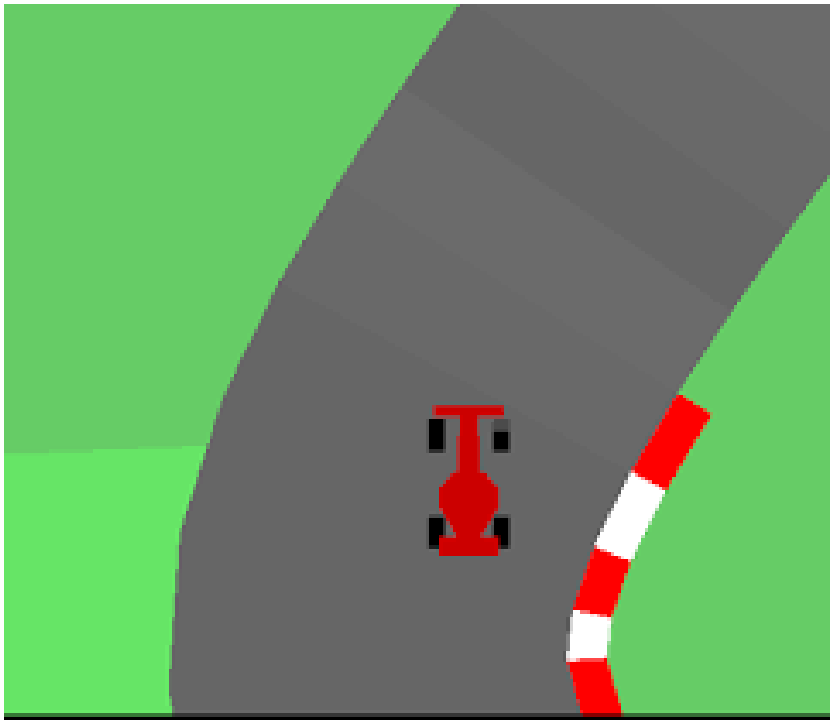

Figura 1: CarRacing-V0

O ambiente disponibiliza uma imagem RGB com resolução de **96x96 pixels**, representando a visão do carro sobre a pista, e o agente precisa interpretar essas imagens para ajustar sua direção corretamente. A pontuação no jogo é baseada na performance do agente, com recompensas positivas atribuídas conforme o carro avança pela pista sem sair dos limites, e penalizações quando o carro sai do percurso.

3 Metodologia

Nossa abordagem consiste em utilizar uma rede neural convolucional (CNN) que, após ser projetada e treinada, terá como entrada os frames capturados do ambiente e como saída os valores discretos correspondentes ao controle. A rede será responsável por controlar o carro de forma contínua, mantendo-o na pista.

Para isso, precisamos desenvolver os seguintes passos:

- Implementar um controlador automático para dirigir o carro, que servirá como "especialista". Esse especialista fornecerá as trajetórias desejadas que a rede neural tentará replicar.
- Desenvolver funções para capturar e salvar os frames gerados pelo ambiente, associando cada imagem ao respectivo valor de controle aplicado naquele momento.
- Desenvolver a arquitetura da rede neural e treiná-la com os dados coletados
- usar a rede treinada para prever os comandos em tempo real e controlar o carro

4 Controle

Para coletar dados de treinamento e servir como "especialista"que a rede neural vai imitar, implementamos um **controlador automático** composto por duas partes principais: o reconhecimento da pista e o controle do carro.

Reconhecimento da Pista (Classe `RecognizesRoad`): Esta classe tem como objetivo identificar a posição do carro na pista com base nas imagens fornecidas pelo simulador. O processo é dividido em três partes principais:

- **Conversão da imagem para binário**: Transformamos a imagem RGB do simulador em uma matriz binária, onde a pista é representada por 1 (preto) e as áreas fora dela por 0 (branco). Essa simplificação facilita a identificação da pista e a posição do carro em relação a ela.
- **Recorte da imagem**: Para focar na área mais importante, seleciona-se uma parte específica da imagem, uma linha de pixels, que mostra a pista um pouco à frente do carro.
- **Cálculo da posição ponderada**: A posição do carro na pista é calculada a partir de uma média ponderada das informações da imagem recortada. Cada ponto do vetor binário da pista recebe um peso de acordo com sua posição horizontal, indicando se o carro está mais à esquerda, à direita ou centralizado, permitindo assim ajustar a direção adequadamente.

Controle do Veículo (Classe `Control`): Esta classe implementa um controlador responsável por ajustar a direção do carro com base no erro calculado a partir da posição do veículo na pista. O controle pode ser realizado através de diferentes abordagens, como PI, PID ou On-Off.

Para o treinamento da rede neural, utilizamos o controle On-Off, que funciona da seguinte forma:

- **Controle On-Off**: Com base no valor da posição, que varia de -100 a +100, determinamos o comando de direção. Se a posição for maior ou igual a 35, o carro vira para a esquerda. Se for menor ou igual a -35, o carro vira para a direita. Para valores entre -34 e 34, o carro segue em linha reta.

5 Coleta de dados

Para organizar os dados de treinamento, cada frame capturado foi salvo individualmente como um arquivo de imagem com nome sequencial, por exemplo, `imagem_0.png`, `imagem_1.png`, e assim por diante. Em paralelo, criamos um arquivo `.csv` que associa cada imagem aos comandos de controle correspondentes. Cada linha do arquivo `.csv` contém o nome da imagem seguido pelos valores de controle. Esses valores estão organizados em duas colunas, A primeira representa a direção (1 para esquerda, 0 para reto, e 1 para direita) e a segunda a aceleracao do veículo (1 para acelerar, -1 para frear e 0 sem aceleração). A estrutura do arquivo `.csv` segue o seguinte formato:

```
imagem_0, 0, 1.0
imagem_1, 0, 1.0
imagem_2, 0, 1.0
```

Durante a coleta de dados para treinar a rede neural, notamos que o jogo CarRacing tem mais retas do que curvas, sendo a maioria das curvas para a esquerda. Para evitar um viés no treinamento, implementamos um contador que garante a mesma quantidade de imagens para cada estado: "curva para a esquerda", "curva para a direita"e "reta". Além disso, é possível definir o número total de imagens a serem coletadas. Para mitigar o risco de overfitting, adotamos uma estratégia de adição de ruído nos dados, dividida em duas frentes:

- **Recorte Aleatório**: Reduzimos os frames capturados de 96x96 pixels para uma dimensão menor de 84x84 pixels. Esse recorte é realizado aleatoriamente nas bordas laterais, bem como no topo e na parte inferior da imagem.
- **Espelhamento**: Para cada frame, invertemos a imagem no eixo horizontal e também ajustamos o sinal de controle de direção. Dessa forma, dobramos o conjunto de dados com o mesmo tempo de coleta

6 Rede Neural

Na estrutura da nossa rede neural, utilizamos uma arquitetura com 2 *outputs* diferentes. Ela consistia em um *input* de 84x84x1, uma imagem em escala de cinza. Em seguida, tínhamos uma convolução com *kernel* de 8x8, 32 *feature maps* e *stride* de 4. Depois, aplicamos um *max pooling* de 2x2, seguido de uma convolução com *kernel* de 4x4, *stride* de 2 e 64 *feature maps*. Em seguida, realizamos uma convolução com *kernel* de 3x3 e *stride* de 2, finalizando com uma rede *fully connected* com 128 neurônios, todas utilizando a função de ativação *ReLU*. No final, tentamos utilizar 2 neurônios com ativação *tanh* e uma função *softmax* com 6 *outputs* (desconsiderando que iríamos frear).

Quando utilizamos a saída como *tanh*, empregamos a função *mean squared error* como função de custo. Já quando usamos *softmax*, utilizamos a *categorical cross entropy*. Como otimizador, utilizamos o *Adam*, com o *learning rate* de 0.001.

Tabela 1: Estrutura da Rede Neural - Output Tanh

Camada	**Tipo**	**Filtro/Kernel**	**Stride**	**Ativação**	**Saída**	**Feature Maps**
1	Input	84x84x1	-	-	84x84x1	1
2	Conv2D	8x8	4x4	ReLU	20x20x32	32
3	Conv2D	4x4	2x2	ReLU	9x9x64	64
4	Conv2D	3x3	2x2	ReLU	4x4x128	128
5	Flatten	-	-	-	2048	-
6	Dense	-	-	ReLU	128	-
7	Dense	-	-	ReLU	128	-
8	Dense	-	-	Tanh	2	-

Tabela 2: Estrutura da Rede Neural - Output Softmax

Camada	**Tipo**	**Filtro/Kernel**	**Stride**	**Ativação**	**Saída**	**Feature Maps**
1	Input	84x84x1	-	-	84x84x1	1
2	Conv2D	8x8	4x4	ReLU	20x20x32	32
3	Conv2D	4x4	2x2	ReLU	9x9x64	64
4	Conv2D	3x3	2x2	ReLU	4x4x128	128
5	Flatten	-	-	-	2048	-
6	Dense	-	-	ReLU	128	-
7	Dense	-	-	ReLU	128	-
8	Dense	-	-	Softmax	6	-

Posteriormente, também testamos o uso de estruturas de dados já treinadas, adicionando uma camada fully connected com os mesmos outputs. Nesses testes, utilizamos MobileNet, ResNet50 e EfficientNet.

7 Resultados

Nossos resultados não foram satisfatórios. Ao treinar nossa rede neural utilizando diferentes quantidades de imagens, acabamos obtendo overfitting, o que impediu que a rede neural conseguisse controlar o carro em tempo real. A acurácia no conjunto de treinamento chegava a 100% em ambas as redes neurais, porém, no conjunto de validação, ela diminuía, ficando entre 25% e 35%.

Utilizando um banco de dados com 3000 imagens, nossa acurácia convergia para 100% em cerca de 100 épocas. Ao aumentar o banco de dados para 9000 imagens, a convergência ocorreu em cerca de 600 épocas. Já na rede neural utilizando o modelo EfficientNet com 9000 imagens, a acuracia nao convergiu.

Tabela 3: Layers of MobileNet architecture

Type / Stride	**Filter Shape**	**Input Size**
Conv / s2	3 x 3 x 3 x 32	224 x 224 x 3
Conv dw / s1	3 x 3 x 32 dw	112 x 112 x 32
Conv / s1	1 x 1 x 32 x 64	112 x 112 x 32
Conv dw / s2	3 x 3 x 64 dw	112 x 112 x 64
Conv / s1	1 x 1 x 64 x 128	56 x 56 x 64
Conv dw / s1	3 x 3 x 128 dw	56 x 56 x 128
Conv / s2	1 x 1 x 128 x 128	56 x 56 x 128
Conv dw / s1	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 256	28 x 28 x 128
Conv dw / s2	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 256	28 x 28 x 256
Conv dw / s1	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 512	14 x 14 x 256
5x Conv dw / s1	3 x 3 x 512 dw	14 x 14 x 512
Conv / s1	1 x 1 x 512 x 512	14 x 14 x 512
Conv dw / s2	3 x 3 x 512 dw	7 x 7 x 512
Conv / s1	1 x 1 x 512 x 1024	7 x 7 x 512
Conv dw / s1	3 x 3 x 1024 dw	7 x 7 x 1024
Conv / s1	1 x 1 x 1024 x 1024	7 x 7 x 1024
Avg Pool / s1	Pool 7 x 7	7 x 7 x 1024
FC / s1	1024 x 1000	1 x 1 x 1024
Softmax / s1	Classifier	1 x 1 x 1000

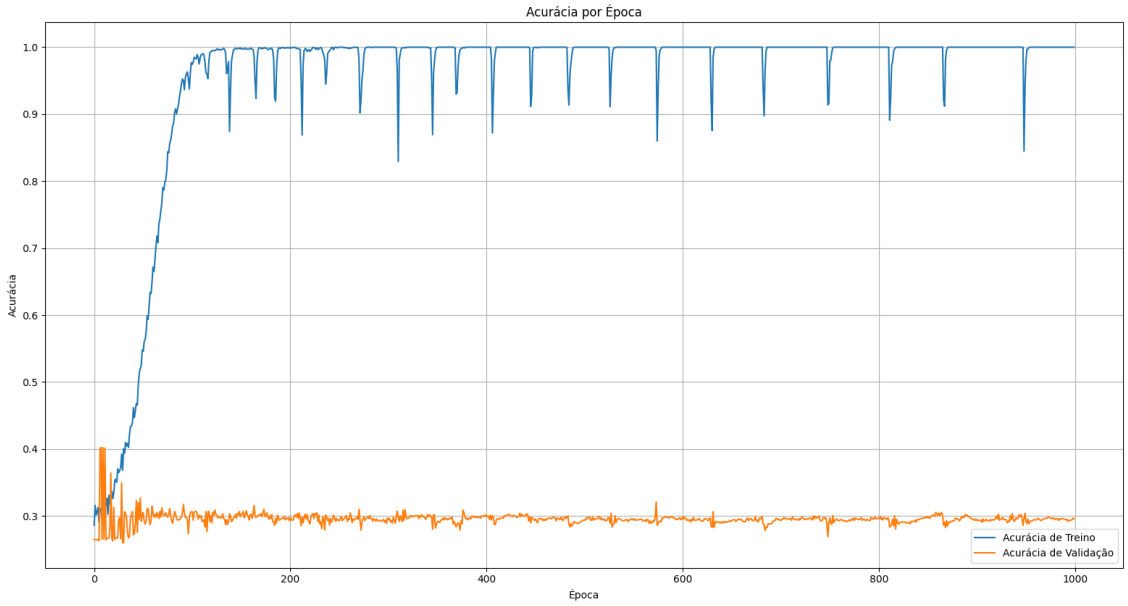

Figura 2: Acurácia com 30006imagens com output softmax

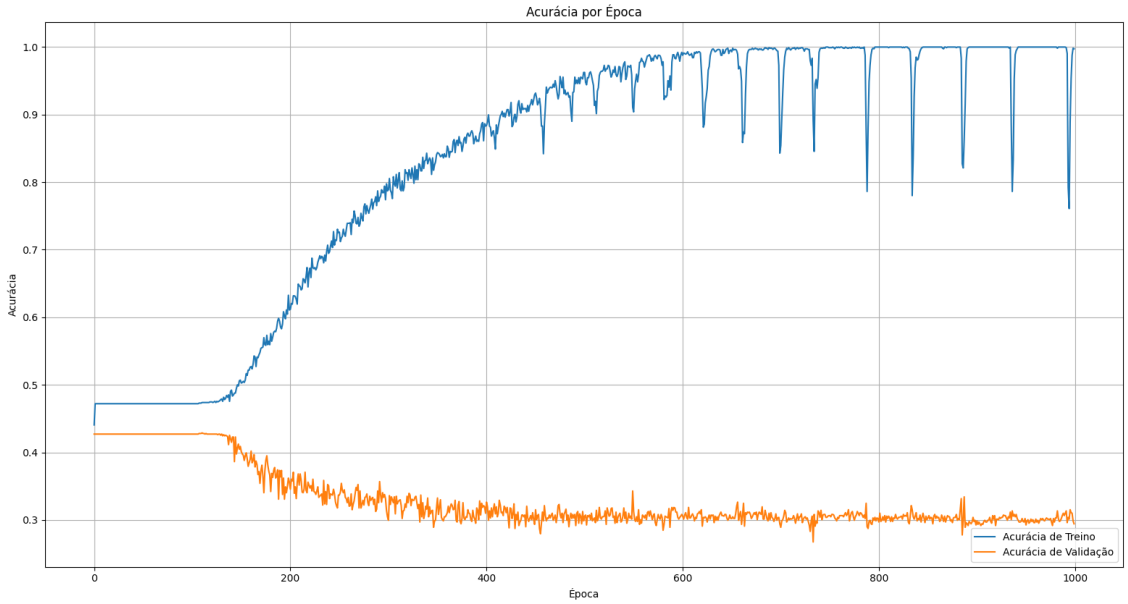

Figura 3: Acurácia com 9000 imagens com output softmax

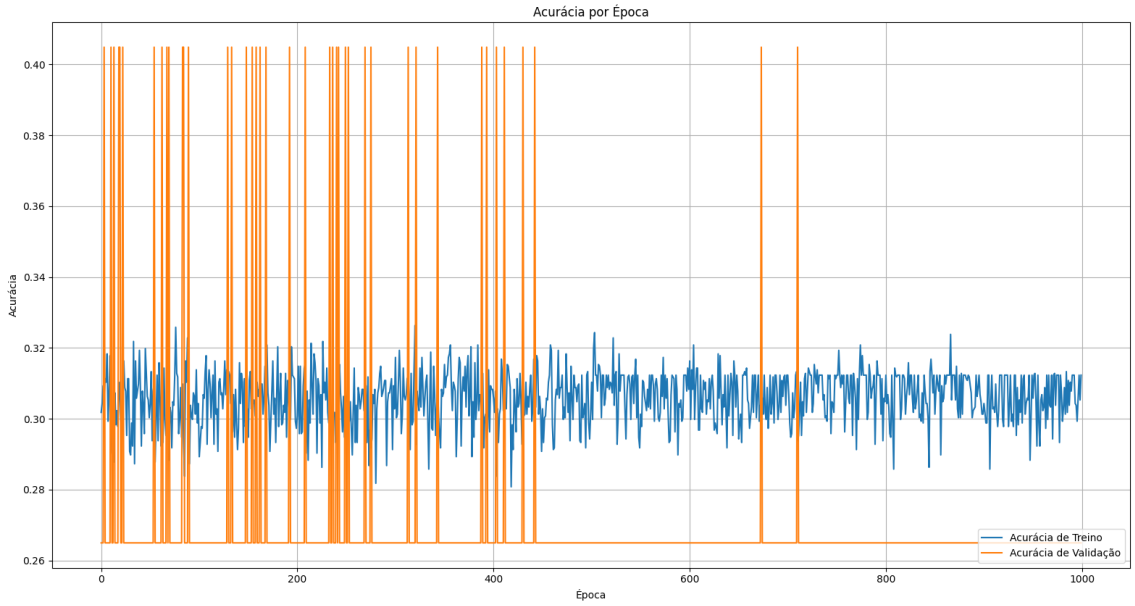

Figura 4: Acurácia com 9000 imagens, EfficientNet e output softmax

Os gráficos acima mostram a acurácia de treinamento e validação para os diferentes tamanhos de banco de dados. Na Figura 4, utilizamos 3000 imagens, enquanto na Figura 2 o banco de dados foi aumentado para 9000 imagens. Embora o aumento de dados tenha estendido o tempo de convergência, ambos os casos sofreram de overfitting.

Pensamos que, talvez, utilizar uma rede neural já treinada pudesse nos ajudar a evitar o overfitting, uma vez que ela já teria uma base para reconhecer formas e alguns padrões. No entanto, obtivemos os mesmos resultados, com a acurácia convergindo no conjunto de treinamento, mas permanecendo em 30% no conjunto de validação.

8 Conclusão

O objetivo deste relatório foi desenvolver um modelo de controle de um carro em ambiente simulado. No entanto, enfrentamos dificuldades significativas relacionadas ao **overfitting**, o que impediu o alcance dos resultados esperados.

Durante o desenvolvimento, o modelo apresentou um desempenho satisfatório durante o treinamento, mas, ao ser testado em novos cenários ou dados, demonstrou limitações severas, sugerindo uma incapacidade de generalizar adequadamente. O **overfitting** foi evidenciado pela alta precisão no conjunto de treinamento e baixo desempenho no conjunto de testes.

Entre as possíveis causas do overfitting, destacamos:

- **Quantidade insuficiente de dados de treino** ou diversidade nos cenários simulados, o que limitou a capacidade do modelo de aprender a se adaptar a diferentes situações.
- **Escolha inadequada de hiperparâmetros**, como uma regularização insuficiente, que pode ter contribuído para o sobreajuste aos dados de treinamento.

Dado esse cenário, alguns pontos importantes para melhorias futuras podem ser sugeridos:

- **Aumento do conjunto de dados**: ampliar a quantidade e a diversidade dos cenários simulados pode ajudar o modelo a generalizar melhor.
- **Otimização do aprendizado por reforço**: explorar diferentes algoritmos e parâmetros de aprendizado por reforço pode facilitar uma melhor interação com o ambiente e promover um aprendizado mais eficaz.

Este trabalho proporcionou um valioso aprendizado no uso de técnicas com TensorFlow, Keras e OpenCV. A experiência adquirida permitiu entender como montar a estrutura de dados para treinar modelos desde o zero, além de como utilizar modelos já existentes e adaptar para nosso problema específico, promovendo uma base para projetos futuros na área de machine learning.

Referências

KERAS. Applications. Disponível em: https://keras.io/api/applications/. Acesso em: 04 out. 2024.

OPENCV. OpenCV Documentation. Disponível em: https://opencv.org/. Acesso em: 04 out. 2024.

VERGOTTEN. Optimizers: A comprehensive study of optimizers in deep learning. Medium, 07 nov. 2019. Disponível em: https://medium.com/@vergotten/optimizers-a-comprehensive-study-of-optimizers-in-deep-learning-80a54490181f. Acesso em: 04 out. 2024.