Universidade Federal de Santa Catarina Centro Tecnológico
Departamento de Automação e Sistemas
Prof. Eric Antonelo
Acadêmica: Lara Popov Zambiasi Bazzi Oberderfer

Relatório: Redes Neurais Convolucionais

2. Treinamento e Avaliação:

   (a) Uma vez escolhido o simulador e agente, defina a arquitetura de rede neural, usando *pyTorch* ou *tensorflow/keras*. Use redes convolucionais para processar entradas do tipo imagem.

   (b) Treine a rede neural com os dados de treinamento. Ao longo do treinamento, avalie seu desempenho plotando a função de custo ao longo do treinamento para os dados de treinamento e para dados de validação. Avalie a rede controlando o agente no simulador.

   (c) Analise os resultados obtidos ao variar os hiperparâmetros da rede neural: número de camadas de convolução/pooling; tamanho do kernel; tamanho do conjunto de treinamento; com ou sem *dropout*, etc. O desempenho do agente piora/melhora? Use métricas.

   (d) Defina um segundo comportamento diferenciado, treine e avalie a rede neural para o mesmo, conforme foi feito para o primeiro comportamento.

O objetivo deste trabalho é utilizar um framework de deep learning para treinar redes neurais profundas a fim de controlar veículos ou robôs através da clonagem comportamental.

Aprendizado por imitação é uma abordagem pela qual um modelo caixa-preta (rede neural) é treinado para imitar um especialista usando um conjunto fixo de amostras de pares observação-ação (ou trajetórias) obtidas daquele especialista.

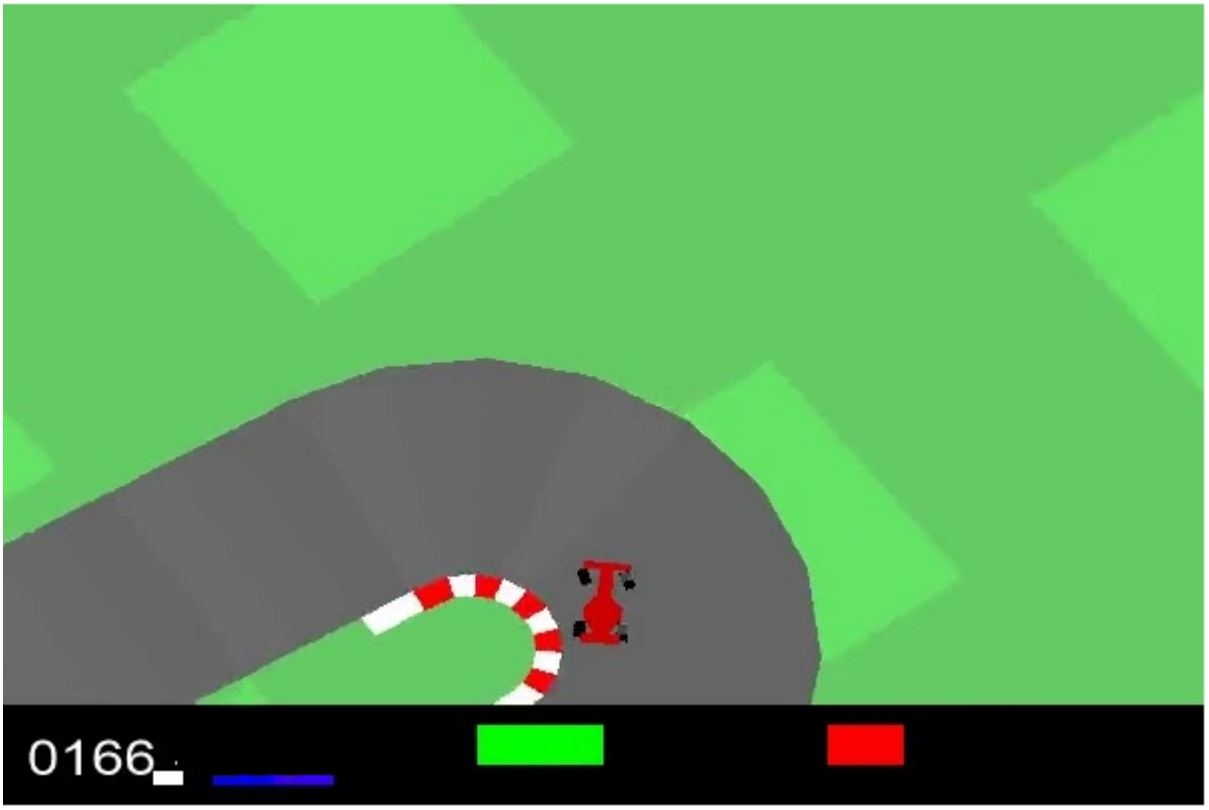
A **clonagem comportamental** (CC) é um tipo de aprendizagem por imitação baseada em um processo de treinamento supervisionado de um modelo (rede neural) usando um grande conjunto de dados rotulados. A CC tem sido utilizada para a obtenção de políticas de condução autônoma para veículos, onde as amostras de treinamento são geradas por motoristas humanos: a entrada da rede neural é a imagem da câmera do carro, enquanto a saída desejada corresponde ao atuador (ação do motorista: aceleração, direção, freio).

Sumário

1. Implementação

O simulador escolhido foi o CarRacing-v0.

Disponível em: https://gym.openai.com/envs/CarRacing-v0/



Este simulador pertence a “OpenAI Gym” é um framework para desenvolver e comparar algoritmos de aprendizagem por reforço. A Gym é uma biblioteca de código aberto, que dá acesso a um conjunto padronizado de ambientes.

Para a implementação do modelo foram utilizadas bibliotecas do PyTorch.

Arquivos implementados para o projeto:

- corrida.py: simulador disponível em: https://gym.openai.com/envs/CarRacing-v0/

- model.py: Rede neural convolucional (existem dois modelos disponíveis, com 4 e 5 camadas)

- gera_lotes.py: arquivo auxiliar de gerar lotes

- grava_imagens.py: grava imagens e arquivo com rotulos

```
   Para gravar uma corrida manualmente para armazenar o lote de
imagens e rótulos
   Digite: ```python3 grava_imagens.py pastaX```

- treino.py: treinamento da rede neural a partir de lote de
imagens
   Para treinar a rede neural convolucional
   Digite: ```python3 treino.py treinoX```

- autonomo.py: arquivo que executa o carro autonomo
   Para executar o teste da rede neural (a pasta e o treino podem
variar)
   Digite: ```python3 autonomo.py treinoX/treinoX_XX.pt```
```

Explicando o funcionamento

Para gerar as imagens (dataset) e o arquivo de rótulos foi utilizada a biblioteca OS e IMAGEIO. O processo é gerado manualmente por meio do simulador. Para salvar as imagens e o arquivo de rotulos.txt durante o processo digitar:

```python3 grava_imagens.py pastaX```

Durante o processo as imagens são armazenadas em uma pasta “imagens”, dentro desta pasta passada pelo comando acima, com o nome img-X.jpg. E o arquivo de rótulos armazena, linha a linha, o nome da imagem img-X.jpg com o rótulo, sendo {0 - esquerda, 1 - direita, 2 - aceleração}.

Para o treinamento precisou-se definir algumas formas de tratamento das informações, tais como geração dos lotes, a função de custo, o modelo de rede convolucional, o otimizador, a forma de salvar o modelo, a forma de validar o custo do treinamento, a forma de avaliar o desempenho da rede neural e a validação dessa medição.

Para gerar os lotes de treinamento e de validação de imagens pode-se definir o tamanho, podem ser pequenos ou grandes, em geral, sugere-se utilizar mini-lotes de pelo menos 32 imagens, carrega-se cada imagem e seu respectivo rótulo, transforma-se essa imagem em formato de tensor e manteve-se o padrão de cores RGB, para isso utilizamos as bibliotecas do TORCH e PIL.

Para a função de custo escolhemos a entropia cruzada (CrossEntropyLoss()) pois, evita o problema de desaceleração do aprendizado. Esta função de perda calcula a diferença entre duas distribuições de probabilidade para um conjunto fornecido de ocorrências ou variáveis aleatórias. Muito utilizada para tarefas de classificação binária, para as quais é a função de perda padrão no Pytorch e para criação de modelos confiáveis, a previsão será precisa e com maior probabilidade.

Para este trabalho utilizou-se dois modelos de redes neurais convolucionais, com alterações em seus hiperparâmetros para os treinos.
```

Rede Neural Convolucional com 4 camadas:

```
Net2 (
  (conv1): Conv2d(3, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(20, 40, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=9000, out_features=100, bias=True)
  (out): Linear(in_features=100, out_features=3, bias=True)
)
```

Esta rede neural conta com duas camadas convolucionais com 5 kernels, a primeira camada tem 3 entradas e 20 saídas, a segunda camada convolucional tem 20 entradas e 40 saídas, a terceira camada é linear e contém 9000 features de entrada e 100 de saída e a última camada contém 100 features de entrada e 3 de saída. Essa rede conta com ativações do tipo RELU e camadas Max Pooling para condensar a figura.

Rede Neural Convolucional com 5 camadas:

```
Net(
  (conv1): Conv2d(3, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(20, 40, kernel_size=(5, 5), stride=(1, 1))
  (dropout1): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=9000, out_features=1000, bias=True)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=1000, out_features=1000, bias=True)
  (dropout3): Dropout(p=0.5, inplace=False)
  (out): Linear(in_features=1000, out_features=3, bias=True)
)
```

O outro modelo de Rede Neural Convolucional utilizado com 5 camadas e 3 dropouts, conta com, 2 camadas convolucionais com 5 kernels, a primeira camada com 3 entradas e 20 saídas, a segunda camada com 20 entradas e 40 saídas, a terceira camada linear com 9000 features de entrada e 1000 de saída, a quarta camada com 1000 features de entrada e 1000 de saída e a última camada com 1000 features de entrada e 3 de saída, cada um dos dropouts foram inseridos com 50% para evitar o overfitting. A rede conta com camadas de ativação RELU e Max Pooling.

Para treinar a rede neural convolucional digite:

```
```python3 treino.py treinoX```
```

Onde, “treinoX”, será a pasta onde os modelos treinados serão guardados.

Após o treinamento de todas as épocas, são plotados os gráficos de Custo e validação ao longo do treinamento por época e outro de Acurácia e validação ao longo do treinamento por época.

Finalmente pode-se verificar qual modelo foi salvo por último, pois o programa só salva os modelos válidos e que tem melhor acurácia.

Para testar o agente na simulação digite:

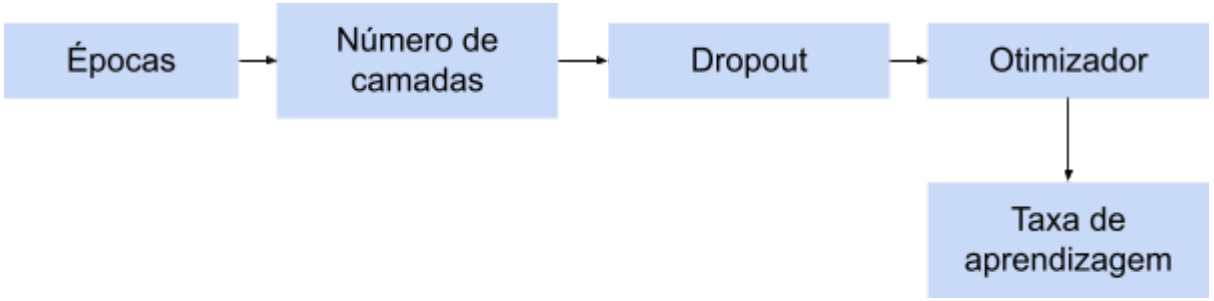
```
```python3 autonomo.py treinoX/treinoXX_X.pt```
```

Onde, “treino”X é a pasta dos modelos treinados e “treinoXX_X.pt” é o último modelo salvo pelo treinamento.

---

2. Metodologia utilizada para o treinamento

Foram realizadas as simulações baseadas no seguinte fluxo de alterações de hiperparâmetros para os treinamentos:



---

3. Resultados obtidos do treinamento da rede

Após a realização de várias simulações com alterações entre duas redes neurais convolucionais, uma com 5 camadas (com dropout) e outra com 4 camadas (sem dropout). Também foram realizadas simulações com dois tipos de otimizadores diferentes, o Adam e o SDG, como também o ajuste de taxa de aprendizado e momentum para maior precisão e foi utilizado lotes (batch) com 32 imagens.

A seguir segue tabela com o resumo dos dados obtidos em cada um dos processos de aprendizado, que estão descritos em detalhes à frente.

Tabela resumo dos dados obtidos referentes aos testes

Treino	Épocas	Otimizador	Taxa de aprendizagem	Tamanho do Kernel	Número de camadas /Dropout	Acurácia	Volta completa
1	20	ADAM	1e-4	5	4 / sem dropout	30%	❌
2	200	ADAM	1e-4	5	4 / sem dropout	31,5	✅
3	1000	ADAM	1e-4	5	4 / sem dropout	31,5%	✅ treino3/
4	100	ADAM	1e-4	5	5 / 0,5	31%	✅
5	20	ADAM	1e-4	5	5 / 0,5	29%	❌
6	20	SDG	1e-4 / momentum = 0.9	5	5 / 0,5	16%	❌
7	100	SDG	1e-4 / momentum = 0.9	5	5 / 0,5	21%	❌
8	30	SDG	0.01 / momentum = 0.3	5	5 / 0,5	30%	❌
9	100	SDG	0.01 / momentum = 0.3	5	5 / 0,5	30%	✅
10	2000	SDG	1e-4/ momentum = 0.3	5	4 / sem dropout	29%	✅ treino10/
11	2000	SDG	1e-4/ momentum = 0.3	5	4 / sem dropout	29,5%	✅ treino11/

Segue descrição dos treinos da “tabela resumo dos dados obtidos referentes aos testes”. Nos treinos estão descritos o otimizador utilizado, a quantidade de épocas, o lote de imagens é de 32 imagens, a taxa de aprendizagem e momentum, foi como forma de métrica utilizou-se os dados da primeira e da última época de treinamento do custo de treinamento e de validação, a acurácia de de cada um deles, a melhor acurácia encontrada. Nos gráficos temos o primeiro gráfico que representa o Custo ao longo do treinamento por época do treino em azul e em laranja da validação e no segundo gráfico, a Acuracidade ao longo do treinamento por época, do treino em azul e em laranja a validação.

Treino com otimizador ADAM: clássico descida do gradiente estocástico

Adam (*Adaptive Moment Estimation*) é um algoritmo de otimização que pode ser usado para atualizar os pesos da rede de forma iterativa com base nos dados de treinamento.

Treino 1

Épocas de treinamento = 20 Lote = 32 imagens Taxa de aprendizagem = 1e-4 Otimizador = Adam Função de custo = CrossEntropyLoss() (entropia cruzada) Rede convolucional: 4 camadas

Época 0/19	Época 19/19
custo treinar: 31.8516 Acuracia: 18.3750 custo validar: 28.9976 Acuracia: 20.8125	custo treinar: 6.0410 Acuracia: 29.1875 custo validar: 5.2366 Acuracia: 29.6250

Melhor acurácia: 29.625000

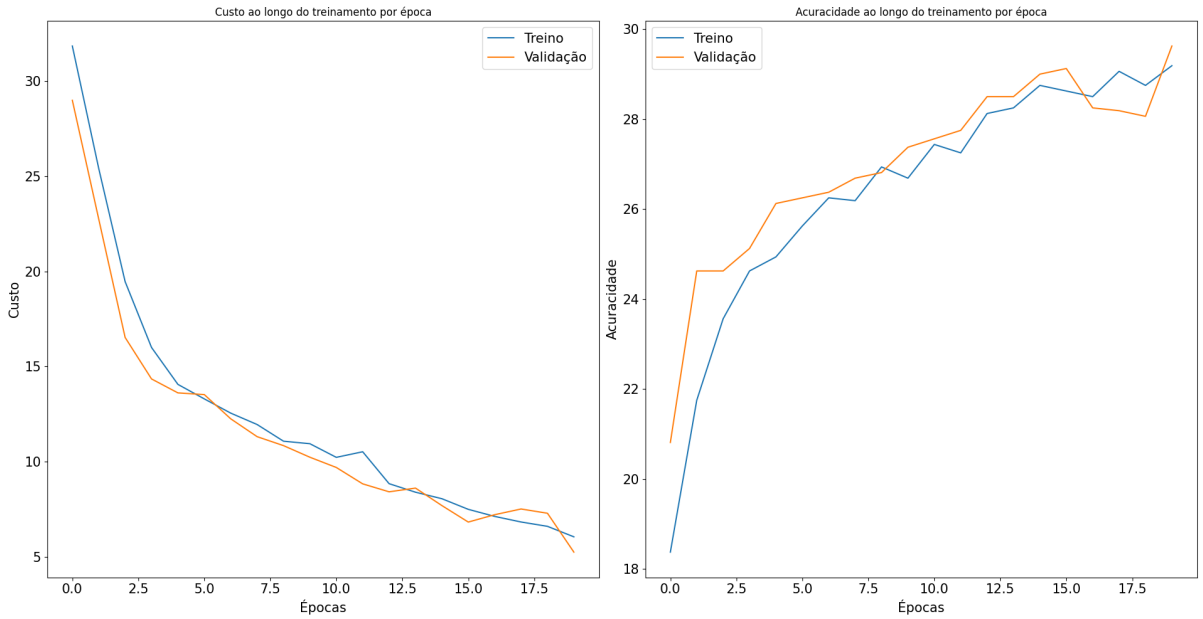


Figura 1: Gráficos do *Custo e Acuracidade do treinamento 20 épocas do* treino 1

Neste treinamento, para a rede neural convolucional utilizou-se duas camadas de convoluções, duas camadas lineares, sem dropouts, com 5 kernels.

O treinamento 1 utilizou o otimizador ADAM, a função de custo de entropia cruzada, a taxa de treinamento 1e-4 e 20 épocas de treinamento.

Como podemos observar na Figura 1, o gráfico de custo utilizando ADAM, tanto de treinamento quanto de validação decresce rapidamente porém não se estabiliza com poucas épocas de treinamento.

Treino 2

Como esse treinamento apresentava possibilidades positivas, foi realizado o treinamento para 200 épocas:

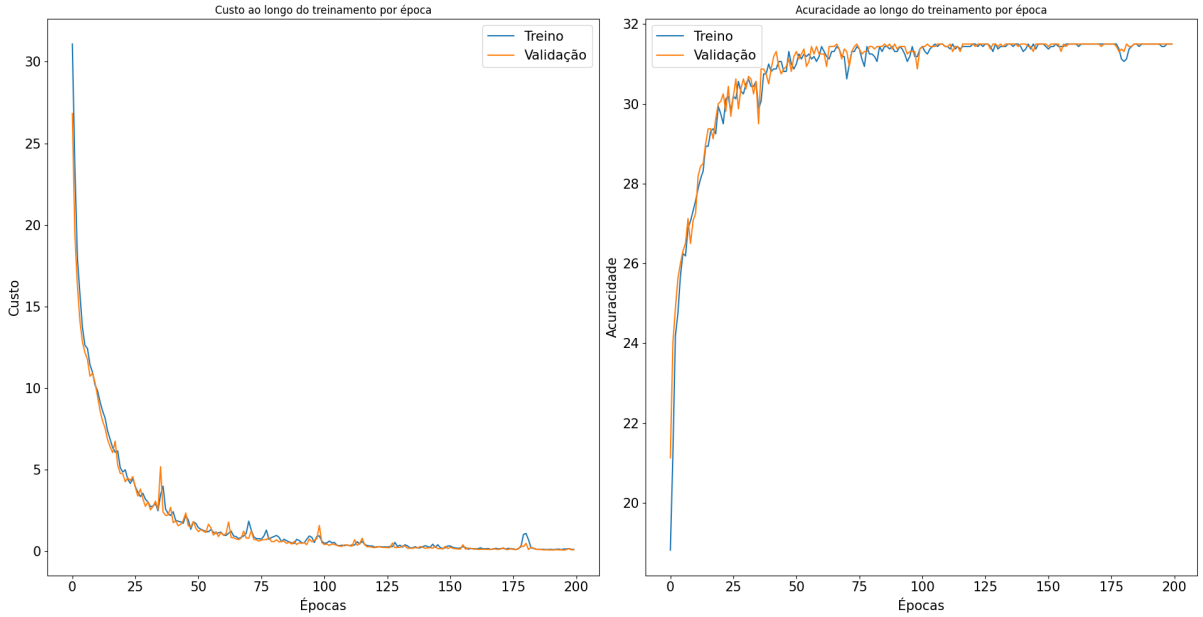


Figura 2: Gráficos do *Custo e Acuracidade do treinamento 200 épocas do* treino 2

Neste treinamento, para a rede neural convolucional utilizou-se duas camadas de convoluções, duas camadas lineares, sem dropouts, com 5 kernels.

O treinamento 1 utilizou o otimizador ADAM, a função de custo de entropia cruzada, a taxa de treinamento 1e-4 e 20 épocas de treinamento.

Como podemos observar na Figura 1, o gráfico de custo utilizando ADAM, tanto de treinamento quanto de validação decresce rapidamente porém não se estabiliza com poucas épocas de treinamento.

Treino 3

Épocas de treinamento = 1000 Lote = 30 imagens Taxa de aprendizagem = 1e-4 Otimizador = Adam Função de custo = CrossEntropyLoss() (entropia cruzada) Rede convolucional: 4 camadas	
Época 0/999	Época 999/999
custo treinar: 32.3834 Acuracia: 19.3125 custo validar: 29.4074 Acuracia: 21.1875	custo treinar: 0.0002 Acuracia: 31.500 custo validar: 0.0001 Acuracia: 31.500
Melhor acurácia: 31.500	

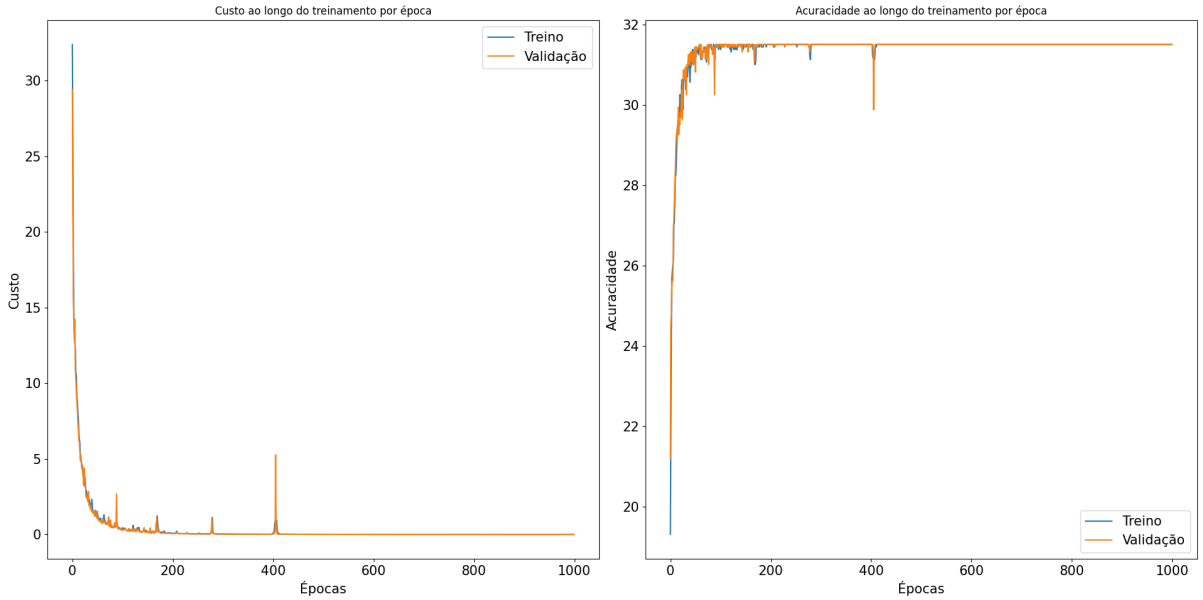


Figura 3: Gráficos do *Custo e Acuracidade do treinamento 1000 épocas do* treino 3

Pode-se perceber pelo gráfico que em aproximadamente 50/60 épocas o custo de treinamento tende a zero. Depois dessa época não há mais aprendizado, como poderemos ver nas análises do capítulo 4.

Treino 4

Como esse treinamento apresentava possibilidades positivas, foi realizado o treinamento para 100 épocas:

Épocas de treinamento = 20 Lote = 32 imagens Taxa de aprendizagem = 1e-4 Otimizador = Adam Função de custo = CrossEntropyLoss() (entropia cruzada) Rede convolucional: 5 camadas

Época 0/99	Época 1/99
custo treinar: 32.0055 Acuracia: 17.6250 custo validar: 27.9601 Acuracia: 20.8750	custo treinar: 0.5788 Acuracia: 31.3125 custo validar: 0.3064 Acuracia: 31.4375
Melhor acurácia: 31.5	

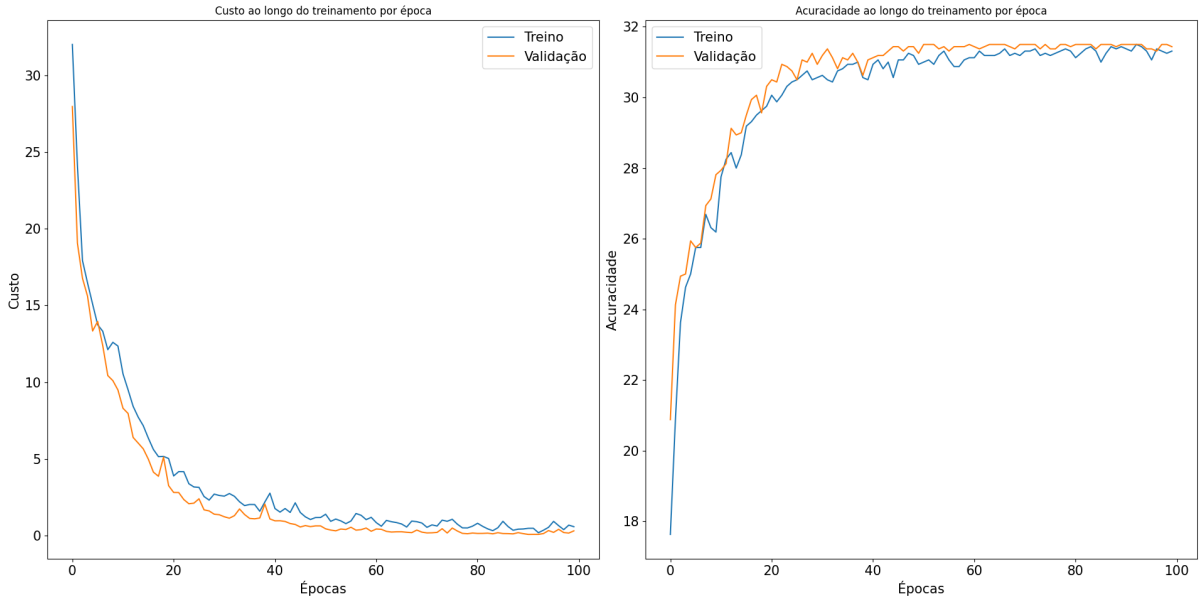


Figura 4: Gráficos do *Custo e Acuracidade do treinamento 100 épocas do* treino 4

Ao longo do treinamento verifica-se que a acurácia das respostas da rede com as imagens e tivemos uma melhor acurácia de 98% ao longo do treinamento.

Com o modelo treinado do lote 50 o carro autônomo consegue dar uma volta inteira.

Treino 5

Épocas de treinamento = 20 Lote = 32 imagens Taxa de aprendizagem = 1e-4 Otimizador = Adam Função de custo = CrossEntropyLoss() (entropia cruzada) Rede convolucional: 5 camadas

Época 0/99	Época 1/99
custo treinar: 32.5690 Acuracia: 15.5000 custo validar: 29.4074 Acuracia: 21.1250	custo treinar: 4.0257 Acuracia: 29.8125 custo validar: 2.6711 Acuracia: 30.6250
Melhor acurácia: 30.62	

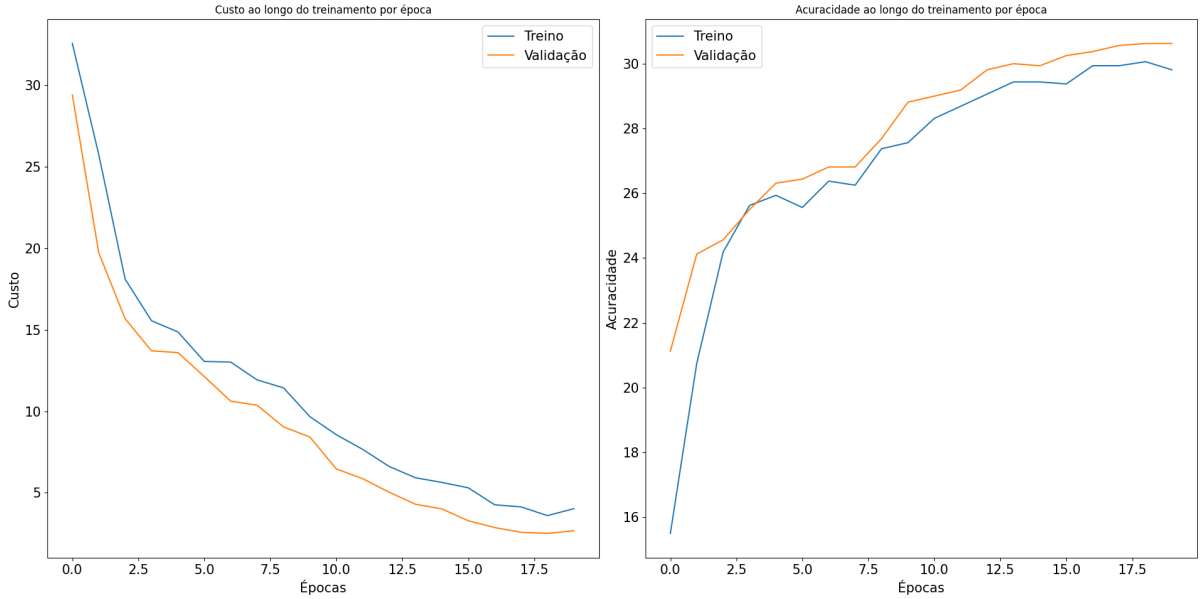


Figura 5: Gráficos do *Custo e Acuracidade do treinamento 20 épocas do* treino 5

Neste treinamento, para a rede neural convolucional utilizou-se duas camadas de convoluções, três camadas lineares, com 3 dropouts de 0,5 cada, com 5 kernels.

O treinamento 1 utilizou o otimizador ADAM (*Adaptive Moment Estimation*), a função de custo de entropia cruzada, a taxa de treinamento 1e-4 e 20 épocas de treinamento.

Treino com otimizador descendente de gradiente estocástico SGD

No gradiente descendente estocástico (SDG - *Stochastic Gradient Descent*), algumas amostras são selecionadas aleatoriamente em vez de todo o conjunto de dados para cada iteração.

Treino 6

Épocas de treinamento = 20 Lote = 32 imagens Taxa de aprendizagem = 1e-4 Otimizador = SGD Função de custo = CrossEntropyLoss() (entropia cruzada) Rede convolucional: 5 camadas	
Época 0/20	Época 19/19
custo treinar: 34.5904 Acuracia: 11.4375 custo validar: 34.5260 Acuracia: 12.5625	custo treinar: 34.0836 Acuracia: 14.3750 custo validar: 33.9940 Acuracia: 16.8125
Melhor acurácia: 16.812500	

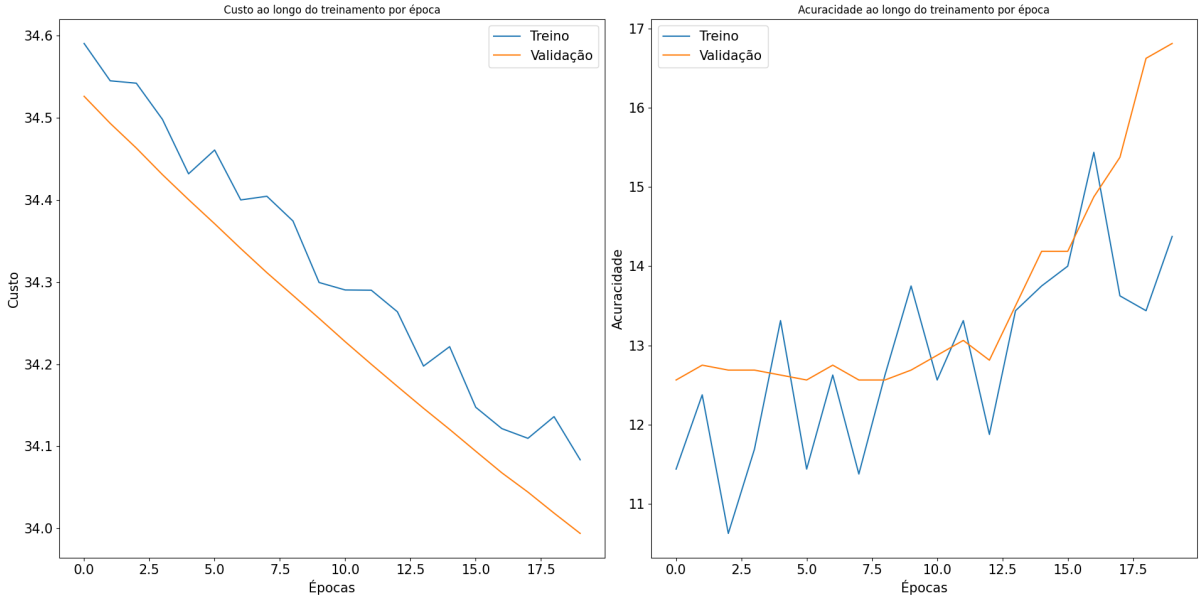


Figura 6: Gráficos do *Custo e Acuracidade do treinamento 20 épocas do* treino 6

Acrescentamos no otimizador SDG o hiperparâmetro momentum = 0.09, que auxilia a saber qual a direção do próximo passo com o conhecimento dos passos anteriores, evitando as oscilações que aparecem no gráfico.

Treino 7

Épocas de treinamento = 100 Lote = 32 imagens Taxa de aprendizagem = 1e-4 Otimizador = SGD, momentum = 0.09 Função de custo = CrossEntropyLoss() (entropia cruzada) Rede convolucional: 5 camadas

Época 0/99	Época 99/99
custo treinar: 34.7849 Acuracia: 8.0625 custo validar: 34.7258 Acuracia: 6.5000	custo treinar: 30.7290 Acuracia: 19.1875 custo validar: 29.9203 Acuracia: 20.5625
Melhor acurácia: 21.1875	

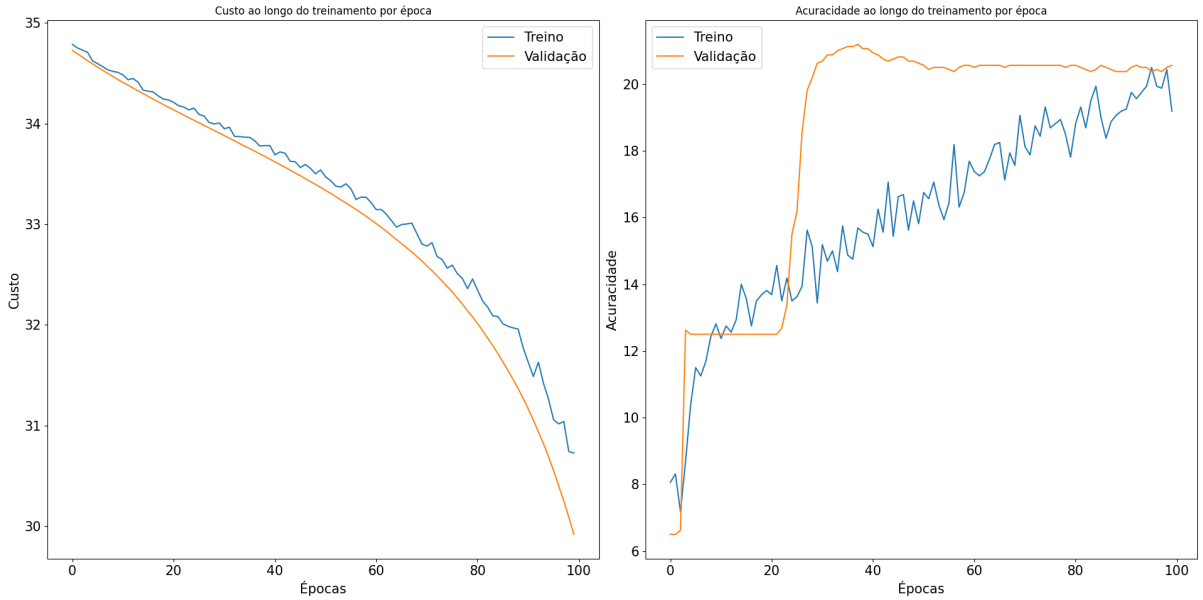


Figura 7: Gráficos do *Custo e Acuracidade do treinamento 100 épocas do* treino 7

Percebemos que com 100 épocas de treinamento com o otimizador descendente de gradiente estocástico (SGD), a curva demora muito para treinar.

Treino 8

Alguns autores sugerem que “O intervalo de valores a considerar para a taxa de aprendizagem é menor que 1,0 e maior que 10 ^ -6. Taxas de aprendizado menores exigirão mais períodos de treinamento. Por outro lado, maiores taxas de aprendizagem exigirão menos períodos de treinamento. Além disso, tamanhos de lote menores são mais adequados para taxas de aprendizado menores, dada a estimativa ruidosa do gradiente de erro” (BENGIO, 2012) .

Sendo assim, ajustando os hiperparâmetros: taxa de aprendizado para 0.01 e ajustando novamente o momentum para 0.3 em 30 épocas, percebeu-se que ao ajustar a taxa do momentum houve uma redução nas oscilações existentes no custo de treinamento e validação.

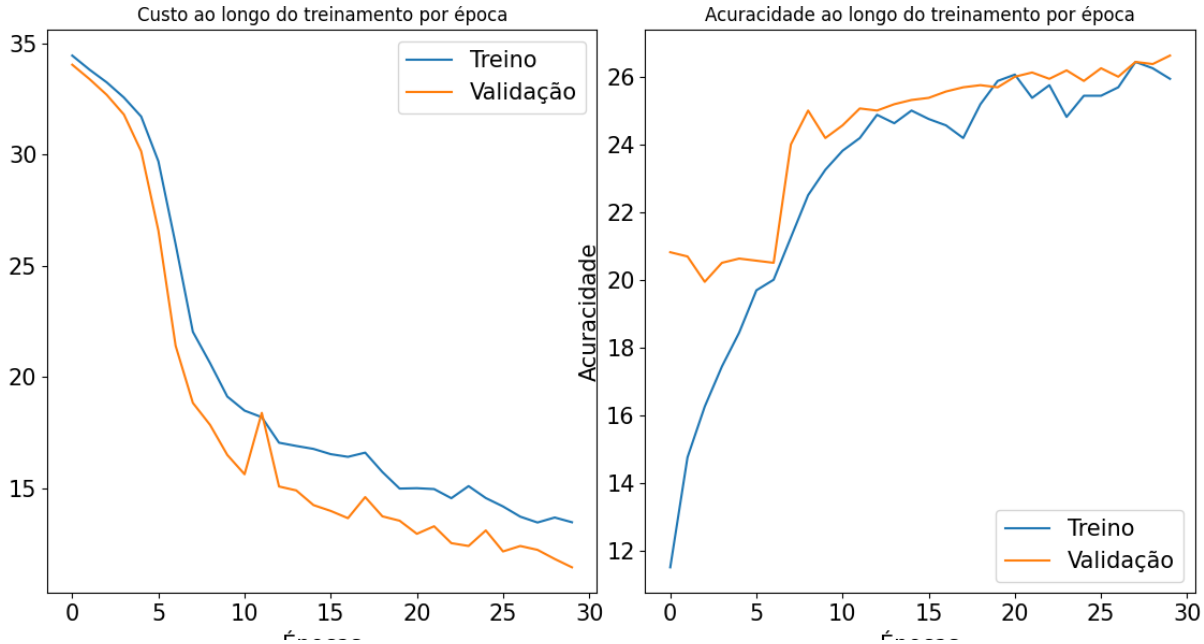


Figura 8: Gráficos do *Custo e Acuracidade do treinamento 30 épocas do* treino 8

Não conseguimos perceber em 30 épocas o otimizador SDG, então treinamos novamente com 100 épocas.

Treino 9

Épocas de treinamento = 100 Lote = 32 imagens Taxa de aprendizagem = 0.01 Otimizador = SGD momentum = 0.3 Função de custo = CrossEntropyLoss() (entropia cruzada) Rede convolucional: 5 camadas	
Época 0/99	Época 99/99
custo treinar: 34.3847 Acuracia: 12.0625 custo validar: 34.0912 Acuracia: 17.0625	custo treinar: 6.3303 Acuracia: 29.3125 custo validar: 3.5900 Acuracia: 30.312
Melhor acurácia: 30.312500	

Ajustando os hiperparâmetros: taxa de aprendizado para 0.01 e momentum para 0.3 com 100 épocas.

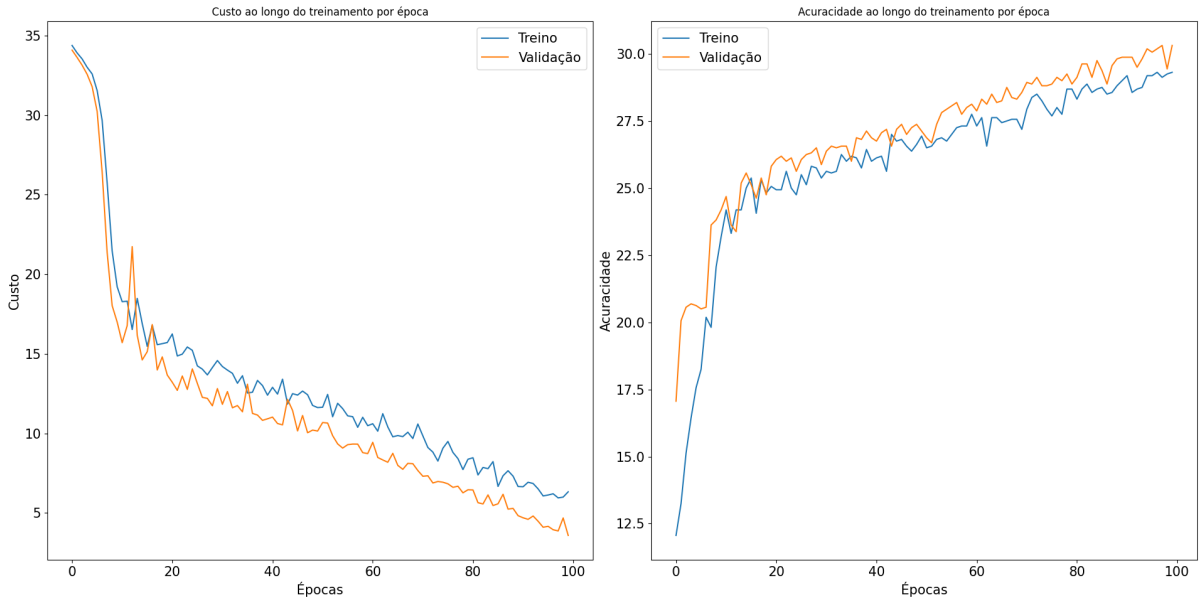


Figura 9: Gráficos do *Custo e Acuracidade do treinamento 100 épocas do* treino 9

Podemos ver nas figuras que, apesar de 100 épocas não serem o suficiente, o custo e a acurácia melhoraram significativamente. E com o modelo treinado do lote 97 o carro autônomo consegue dar várias voltas na pista, mostrando-se promissor para o aumento de épocas de treinamento.

Treino 10

Épocas de treinamento = 2000 Lote = 32 imagens Taxa de aprendizagem = 1e-4 Otimizador = SGD momentum = 0.3 Função de custo = CrossEntropyLoss() (entropia cruzada) Rede convolucional: 4 camadas

Época 0/1999	Época 1999/1999
custo treinar: 34.6679 Acuracia: 6.5625 custo validar: 34.6545 Acuracia: 6.5625	custo treinar: 6.7790 Acuracia: 29.3750 custo validar: 6.7247 Acuracia: 29.3125
Melhor acurácia: 29.437500	

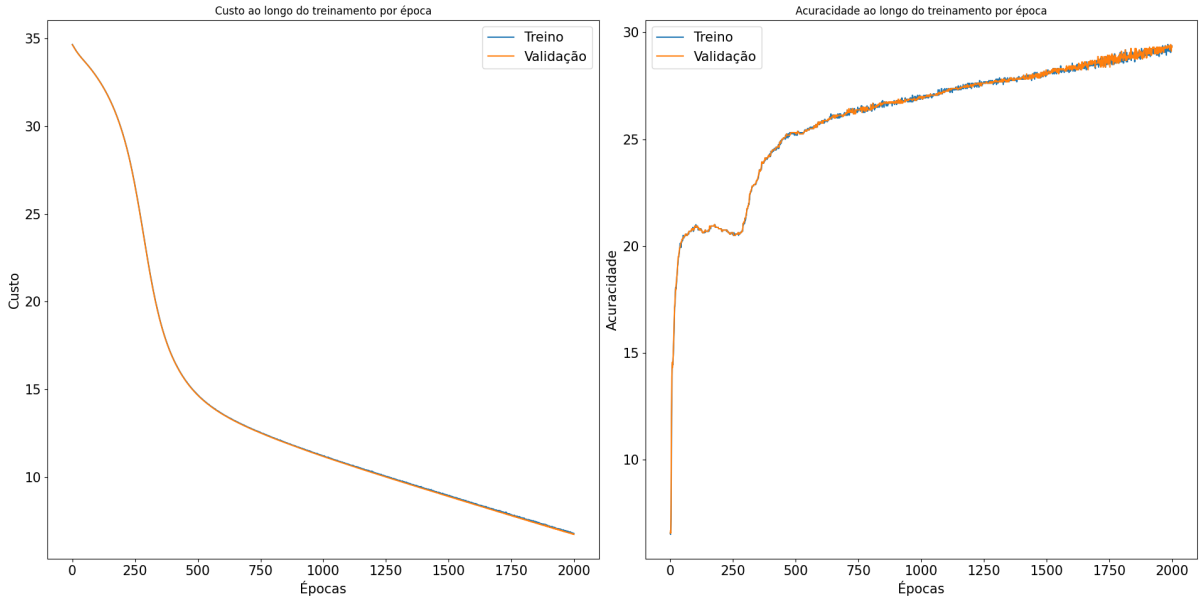


Figura 10: Gráficos do *Custo e Acuracidade do treinamento 2000 épocas do* treino 10

Pode-se observar que no gráfico de acurácia houve dois degraus de estabilização, o primeiro um pouco após as 250 épocas, o que não estava sendo previsto nos outros testes, pois estavam sendo relacionados a 200 épocas, e após houve nova elevação até 500 épocas, e se observou que a acurácia voltou a ter uma curva de evolução mais horizontal.

Treino 11 - Com alteração de comportamento

Para esse treino foi realizado um novo comportamento do tipo zig-zag, fora da pista e com rodopio, com um novo grupo de imagens para treino.

Épocas de treinamento = 2000 Lote = 32 imagens Taxa de aprendizagem = 1e-4 Otimizador = SGD momentum = 0.3 Função de custo = CrossEntropyLoss() (entropia cruzada) Rede convolucional: 4 camadas	
Época 0/1999	Época 1999/1999
custo treinar: 34.6679 Acuracia: 6.5625 custo validar: 34.6545 Acuracia: 6.5625	custo treinar: 6.6400 Acuracia: 29.4375 custo validar: 6.6194 Acuracia: 29.4375
Melhor acurácia: 29.5000	

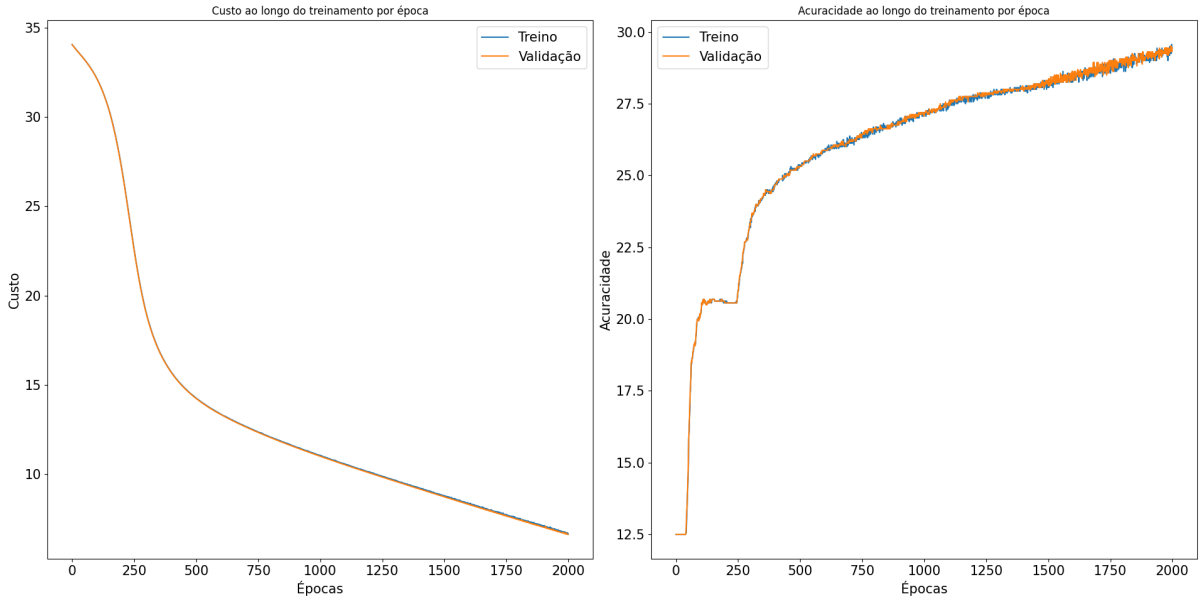


Figura 11: Gráficos do *Custo e Acuracidade do treinamento 2000 épocas do* treino 11

Pode-se observar que no gráfico de acurácia houve dois degraus de estabilização, o primeiro um pouco após as 250 épocas, o que não estava sendo previsto nos outros testes, pois estavam sendo relacionados a 200 épocas, e após houve nova elevação até 500 épocas, e se observou que a acurácia voltou a ter uma curva de evolução mais horizontal.

Observando-se os gráficos de custo dos treinos 10 e 11 são muito semelhantes, mas os gráficos de acurácia tem uma leve diferença.

4. Resultados obtidos do agente

Após os treinamentos com as duas redes neurais convolucionais escolheu-se os modelos dos treinos considerados com melhor acurácia, ou seja, com maiores acertos e os menores falsos negativos e falsos positivos em todas as épocas de validação para medir o desempenho do agente. Para todas as simulações reduzimos a intervenção de aceleração no agente em 50%, para que o mesmo conseguisse completar o percurso.

Modelo do Treino 3 para simulações com o agente (Adam com 1000 épocas)

O treinamento salvou ao total 25 modelos, considerando o de melhor acurácia o modelo 58, isso porque durante o treinamento, após a época 58 não encontrou nenhum modelo com acurácia maior que esta para salvar, observando o gráfico do treino 3, isso fica claro, pois a curva do treinamento cresce rapidamente e estabiliza até esta época.

Observando o desempenho do agente (carro autônomo) no simulador, com esse modelo, ele não é um desempenho muito bom, sai bastante da pista, dá bastante giros, como segue a tabela com alguns testes.

Para os testes limitamos a 30 segundos de simulação, avaliamos o total de pontos e fizemos uma média de total de pontos por segundos.

Tabela avaliações do agente no simulador

Simulação	Tempo total	Total pontos	Total pontos/segundos
1	30.01	487.00	16.22
2	30.01	492	16.41
3	30.00	457.57	15.24
4	30.03	-37825.77	-1259
5	30.03	815.91	27.16
6	30.01	-80437.03	-2679.60
7	30.01	-92299.74	-3074.68
8	30.01	-53189.52	-1771.86
9	30.03	789.97	26.30
10	30.01	-34682.35	-1155.43
Média de pontos da simulação			-983.924

Rodando 10 simulações seguidas com o agente, podemos perceber que, em 30 segundos, esse modelo tem 50% de pontuação positiva e 50% de pontuação negativa. E a média total dos 10 treinamentos foi de -983.924 pontos.

Modelo do Treino 10 para simulações com o agente (SDG com 2000 épocas)

O treinamento salvou ao total 162 modelos, considerando o de melhor acurácia o modelo 1994, isso porque o otimizador SGD tem a característica de ir melhorando a acurácia ao longo das épocas, de forma iterativa, para encontrar o mínimo local, então a medida que as épocas vão passando, e a rede neural vai encontrando um novo mínimo local, e é salvo um novo modelo, assim, a última época salva foi a 1994 com a maior acurácia. Observando o gráfico do treino 10, isso fica claro, pois a curva do treinamento cresce lentamente, a estabilização se daria com mais épocas de treinamento, mas pela demora do treinamento não realizamos essa tarefa.

Observando o desempenho do agente (carro autônomo) no simulador, com esse modelo, em todos os treinos foi um ótimo desempenho, sai pouco da pista e faz curvas, como segue a tabela com alguns testes.

Para os testes limitamos a 30 segundos de simulação, avaliamos o total de pontos e fizemos uma média de total de pontos por segundos.

Tabela avaliações do agente no simulador

Simulação	Tempo total	Total pontos	Total pontos/segundos
1	30.01	808.51	26.93
2	30.04	806.24	26.83
3	30.02	780.73	26.00
4	30.04	779.91	25.95
5	30.04	761.60	25.34
6	30.05	772.41	25.69
7	30.04	806.19	26.83
8	30.04	762.19	25.36
9	30.02	772.73	25.73
10	30.03	782.49	26.04
		Média de pontos da simulação	26,07

Rodando 10 simulações seguidas com o agente, podemos perceber que, em 30 segundos, esse modelo tem 100% de pontuações positivas. E a média total dos 10 treinamentos foi de 26,07 pontos.

Modelo do Treino 11 para simulações com o agente (SDG com 2000 épocas)

O treinamento salvou ao total 170 modelos, considerando o de melhor acurácia é o modelo 1993. Observando o gráfico do treino 10, isso fica claro, pois a curva do treinamento cresce lentamente, a estabilização se daria com mais épocas de treinamento, mas pela demora do treinamento não realizamos essa tarefa.

Observando o desempenho do agente (carro autônomo) no simulador, com esse modelo, em todos os treinos foi um ótimo desempenho, sai pouco da pista e faz curvas, como segue a tabela com alguns testes.

Para os testes limitamos a 30 segundos de simulação, avaliamos o total de pontos e fizemos uma média de total de pontos por segundos.

Tabela avaliações do agente no simulador

Simulação	Tempo total	Total pontos	Total pontos/segundos
1	30.04	752.01	25.02
2	30.01	792.68	26.40
3	30.01	759.63	25.30
4	30.04	766.44	25.51
5	30.01	816.99	27.21
6	30.01	817.09	27.21
7	30.04	792.92	26.57
8	30.04	795.71	26.50
9	30.01	799.37	27.63
10	30.01	817.47	27.23
		Média de pontos da simulação	26,45

Rodando 10 simulações seguidas com o agente, podemos perceber que, em 30 segundos, esse modelo tem 100% de pontuações positivas. E a média total dos 10 treinamentos foi de 26,45 pontos.

Comparando o treino 11 com o treino 10, a média de pontos da simulação do treino 11 teve um resultado um pouco melhor que o do treino 10. Por mais que o treino foi criado a partir de um comportamento totalmente errado, com o carrinho em zig-zag, saindo da pista e rodopiando.

---

5. Considerações Finais

Pode-se concluir que, ao realizar mudanças nos hiperparâmetros houve melhora nos resultados obtidos.

Os testes com baixo número de épocas foram importantes para perceber alterações de convergências nos gráficos de custo de treinamento e validação, bem como na evolução da acurácia da rede, para então realizar novos testes alterando hiperparâmetros, com maiores épocas.

Alterações no número de camadas nos modelos testados influenciaram no resultado do aprendizado, concluindo-se que menos camadas resultaram em modelos mais precisos.

Com relação ao uso de otimizadores, vai depender da necessidade, o otimizador ADAM alcança o mínimo global mais rapidamente mas para o nosso caso, o otimizador SDG, com taxa de aprendizagem e momentum adequados foram mais eficazes. O momentum ajuda a saber qual a direção do próximo passo com o conhecimento dos passos anteriores, evitando as oscilações.

Comparando os modelos salvos ao longo dos treinamentos, para a posterior inferência no agente, observou-se que entre os otimizadores Adam e SGD, o primeiro chega a um nível de acurácia ótima no início do treinamento. Já o SGD evolui o valor da acurácia até o fim do seu treinamento, tendo seu ápice entre as últimas épocas.

Também concluiu-se que o comportamento das entradas da simulação podem conter distorções, pois elas não influenciam no resultado da rede neural.

Em conclusão, para que haja bom desempenho de um modelo, no desenvolvimento de uma rede neural convolucional, o ajuste de parâmetros em relação ao número de camadas, quantidade de kernel, otimizadores, taxa de aprendizagem, momentum, dropout, como a quantidade de épocas a serem treinadas irão depender do tipo e quantidade de informação que será interpretada e os resultados esperados.

6. Referências bibliográficas

Bishop, Christopher M. Pattern recognition and machine learning. Springer, 2006.

BENGIO, Y. (2012). Practical recommendations for gradient-based training of deep architectures. Disponível em: https://arxiv.org/abs/1206.5533.

BROWNLEE, J. PyTorch Tutorial: How to Develop Deep Learning Models with Python. Disponível em: https://machinelearningmastery.com/pytorch-tutorial-develop-deep-learning-models/

CECCON, D. (2019) Fundamentos de ML: funções de custo para problemas de regressão. Disponível em: https://iaexpert.academy/2019/08/19/fundamentos-de-ml-funcoes-de-custo-para-problemas-de-regressao/

E-Book: Deep Learning Book. 2021. Disponível em: http://deeplearningbook.com.br/cross-entropy-cost-function/

SALU. (2017) Kaggle Dogs vs. Cats Redux on Floydhub. Disponível em: https://shaoanlu.wordpress.com/2017/04/22/deep-learning-on-floyd-dogs-vs-cats-redux-kaggle-competition/ and https://github.com/shaoanlu/dogs-vs-cats-redux/blob/master/opt_experiment.ipynb

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media, 2009. Disponível em: https://web.stanford.edu/~hastie/Papers/ESLII.pdf

SHARMA, P. (2019) Build an Image Classification Model using Convolutional Neural Networks in PyTorch. Disponível em: https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/

KINGMA, D. & BA, J. (2014) Adam: A Method for Stochastic Optimization. Disponível em: https://arxiv.org/abs/1412.6980

Nested Software. (2019) PyTorch Image Recognition with Convolutional Networks. Disponível em: https://nestedsoftware.com/2019/09/09/pytorch-image-recognition-with-convolutional-networks-4k17.159805.html