Deep Reinforcement Learning Using a Low-Dimensional Observation Filter for Visual Complex Video Game Playing

André M. P. de Quinta[1], **Arthur P. Labaki**[1], **Vinicius C. Rocha**[1], **Vinnicius P. Silva**[1]

[1]Faculdade de Computação – Universidade Federal de Uberlândia (UFU)
Av. João Naves de Ávila, 2121 – 38.400-902 – Uberlândia – MG – Brasil

andre.morais@ufu.br, arthur.labaki@ufu.br,

viniciuscalixto.grad@ufu.br, vinnicius.silva@ufu.br

Resumo. *Este artigo tem como objetivo apresentar a aplicação de técnicas de aprendizado de máquina para o jogo Neon Drive, onde o jogador deve desviar de obstáculos. Devido um problema no código-fonte disponibilizado pelos autores, foi realizado algumas mudanças no objetivo geral do projeto proposto. Para aumentar a eficiência do algoritmo desenvolvido pelos autores, foram implementadas modificações na etapa de pré-processamento de imagem, em relação a limiarização da imagem. Os resultados dessas alterações foram analisados e comparados utilizando diferentes métricas para compreender como a mudança das técnicas afeta o desempenho do algoritmo no desafio proposto.*

1. Introdução

Com o passar dos anos, os jogos eletrônicos têm evoluído significativamente na qualidade gráfica e complexidade graças ao avanço da tecnologia. Como resultado, assuntos relacionados ao pré-processamento de tais imagens complexas se tornaram um contexto de interesse crescente para a inteligência artificial (*IA*), visto que, para que máquinas aprendam em tais cenários, abstrações e simplificações devem ser realizadas. O uso de técnicas de *IA* e filtros de baixo dimensionamento são abordagens eficazes para melhorar a jogabilidade e aumentar a capacidade de uma máquina de vencer e aprender a jogar jogos complexos.

Esse artigo tem o intuito de demonstrar e discutir diferentes técnicas de pré-processamento e os benefícios que elas podem ou não proporcionar ao utiliza-las junto a uma inteligência artificial, tendo como base um artigo selecionado. Além disso, será realizado diversos comparativos entre essas técnicas, a fim de demonstrar quais deles são os mais aconselháveis a serem usados com um tipo especifico de aprendizado de máquina (aprendizado por reforço profundo).

2. Contextualização

De acordo com o artigo mencionado [Kich et al. 2022], o objeto de treinamento do algoritmo de aprendizado de máquina foi o Neon Drive, que é um jogo onde o jogador deve desviar dos obstáculos impostos movimentando-se para o lado esquerdo ou direito. Para isso, é necessário realizar o pré-processamento do vídeo obtido do jogo capturado em tempo real, em que cada *frame* obtido é tratado individualmente, utilizado uma série

de procedimentos para esse tratamento, sendo elas: a conversão para níveis de cinza, o corte manual da região crítica e seu redimensionamento, a utilização de um algoritmo de segmentação de imagem (*thresholding*), a inversão das cores (bits) da imagem processada e o cálculo da diferença temporal da sequencia de *frames* obtidos respectivamente de acordo com a Figura 1.

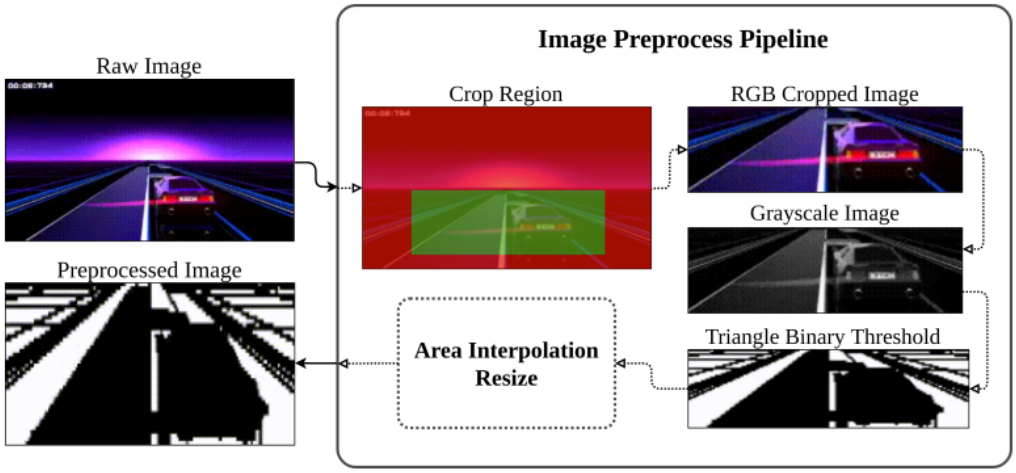

Figura 1. Demonstração das camadas de pré-processamento

Com o pré-processamento realizado, a rede neural convolucional do tipo aprendizado de reforço profundo foi submetida a um processo de treinamento, com o objetivo de tomar decisões de movimentação corretas no jogo selecionado, baseando-se em uma função de recompensa estabelecida previamente, que resume-se em uma pontuação obtida com base no tempo em que o carro avança pelo percurso do jogo sem se colidir com os obstáculos. Após o treinamento, a rede neural foi submetida a um teste de robustez, que é um conjunto de testes para verificar a capacidade de continuar funcionando corretamente mesmo sob condições adversas ou imprevistas, confirmando que o algoritmo treinado realmente aprendeu.

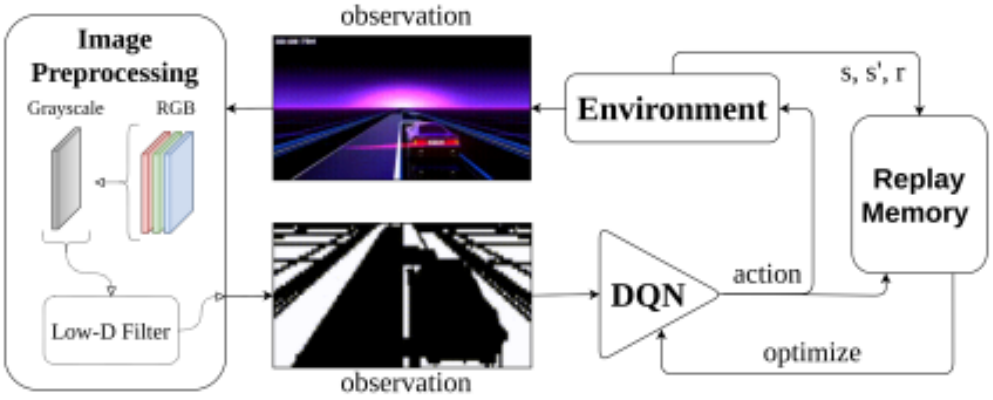

Figura 2. Demonstração de execução do projeto do artigo

O artigo também disponibiliza um repositório público no GitHub com os códigos usado no desenvolvimento do projeto, no qual é explicado, de maneira geral, algumas

informações básicas do projeto, além de demonstrar como realizar a execução do projeto, e como obter os resultados. Também é disponibilizado um video explicativo resumido dos resultados do projeto.

2.1. Contribuições

As contribuições do presente trabalho incluem a implementação de técnicas de pré-processamento de imagens, com o objetivo de aumentar a eficiência das operações realizadas. Em particular, foi proposto o corte dos *frames* antes de iniciar o tratamento, visando reduzir a resolução das imagens e, consequentemente, aumentar a eficiência das operações. Além disso, foram realizadas modificações na função de limiarização, a fim de verificar qual dos tipos de *threshold* apresenta melhores resultados, mediante a utilização de métricas previamente estabelecidas.

A realização desse trabalho visa a implementação de técnicas e didáticas aprendidas durante as aulas de multimídia, através de modificações no código disponível. As modificações propostas buscam aprimorar a eficiência das operações realizadas e contribuir para a melhoria da qualidade dos resultados obtidos.

2.2. Objetivos

Os objetivos previstos para a realização do trabalho foram:

1. Modificar a etapa de pré processamento do artigo e comparar com o mostrado no artigo;
2. Adicionar novas etapa de pré processamento do artigo e comparar com o mostrado no artigo;
3. Utilizar outros ruídos para o teste de robustez a fim de comparar os resultados obtidos entre eles;

Explicando melhor, temos que o primeiro objetivo se trata de modificar a ordem das operações realizadas no pre-processamento da imagem capturada, ou também remover diferentes etapas dela, a fim de verificar e comparar qual delas tem o melhor desempenho comparado com o resultado obtido no artigo. Já o segundo objetivo refere-se em adicionar novas etapas de pré-processamento no código, como uma compressão ou filtros, ou também adicionar outras versões das operações já realizadas, como alterar de *threshold triangle* para *threshold niblack* por exemplo, ainda com o intuito de compara-los com o resultado obtido no artigo.

Por fim, podemos utilizar outros tipos de ruídos testados nos resultados da execução da rede neural, substituindo o ruido *Sal com Pimenta* por o *Gaussiano* ou o *Poisson* por exemplo, tendo como objetivo verificar se os resultados testados são realmente confiáveis.

3. Metodologia

Para tentar resolver os objetivos propostos, a metodologia foi dividida em três principais atividades, a alteração da ordem das camadas de pré-processamento dos *frames*, a variação de diferentes thresholds para obtenção do primeiro frame binário e o treinamento da rede para a contraposição dos resultados obtidos por alterações com os originais propostos pelos autores. Devido a pontos não esperados que são explicados nos subtópicos dessa seção é apresentado primeiro as etapas de treinamento e depois as etapas de variação das camadas e *thresholds* do pré-processamento.

3.1. Treinamento da DQN

Para a etapa inicial do desenvolvimento dos objetivos foram realizados diversos testes de aprendizado incluindo as arquiteturas originais e novas com variações das etapas de pré-processamento, para a possível verificação da corretude da metodologia dos autores do algoritmo seguindo o estabelecido e, também, para contrapor utilizando os novos *thresholds* na etapa de obtenção da imagem.

Embora foi realizado uma boa quantidade de testes, utilizando sistemas operacionais diferentes(ubuntu linux e windows) em maquinas diferentes, a rede convolucional não apresentou melhoras significativas com relação à função de recompensa mesmo com instâncias de treinamento de, na média, 7 horas e meia como o proposto pelos autores. Com isso, o enfoque da metodologia passou a ser relacionado às etapas de pré-processamento da imagem. Abaixo o gráfico de recompensas por episódio considerando o cenário proposto pelos autores sem modificações no código-fonte.

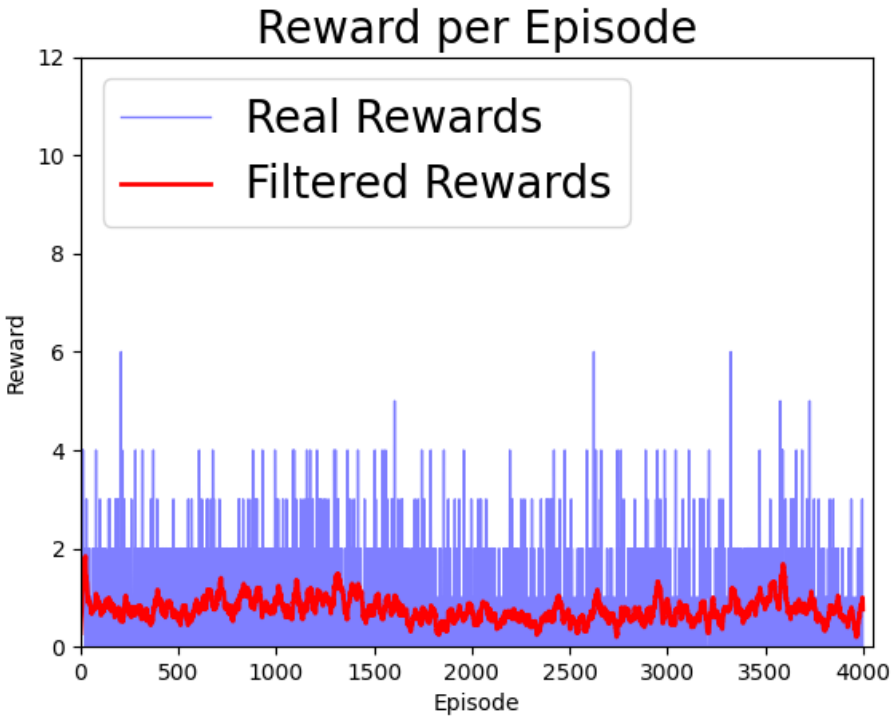

Figura 3. Gráfico de recompensas por episódio da arquitetura original

3.2. Variação da ordem das etapas de pré-processamento

No escopo da etapa de pré-processamento da imagem, duas variações foram as principais, a mudança da ordem utilizada para obter o *frame* pré-processado binário final e a alteração da função de *threshold* que é utilizada para a obtenção do primeiro *frame* binário.

No cenário do treinamento utilizando os *frames* em tempo real, o pré-processamento ideal da imagem é um por *frame* e, portanto, 60 por segundo, com isso, o tempo de execução de uma instância de pré-processamento ideal não deve ultrapassar

de 16 milissegundos. Como a ordem original considera o corte da região crítica após a etapa da obtenção do *threshold*, foi posicionado o corte para antes do *threshold* com a finalidade de ganho de desempenho, uma vez que, a função de obtenção do *threshold* pode ser custosa dependendo do modelo utilizado e diminuir a imagem reduz o tamanho de entrada da função.

3.3. Variação dos *thresholds*

Na etapa foram verificadas para posterior seleção mais de 10 tipos de técnicas de *threshold* e diferentes limiares constantes. Para a seleção de 5 entre eles foi realizada a conversão de um vídeo do Neon Drive utilizando como filtro os tipos de *thresholds* e diferentes constantes e posterior seleção empírica em conjunto dos autores deste trabalho, as técnicas selecionadas foram: thresholds triangle, sauvola, niblack, li e constante 25. Um *frame* dos vídeos com filtros aplicados é mostrado na Figura 4.

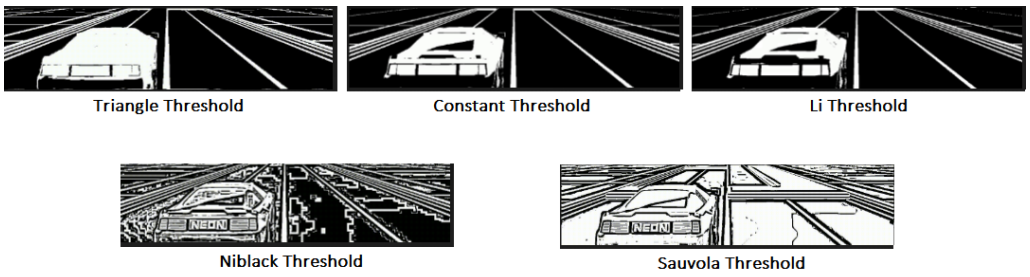

Figura 4. *Frame* dos vídeos com filtros aplicados

O objetivo principal das funções de *thresholds* e para servir de limiar comparativo com o filtro em escalas de cinza do *frame* para a obtenção da primeira imagem binária e para comparar as técnicas selecionadas é aplicado modificações no código-fonte dos autores da versão original. Os autores disponibilizaram dois módulos principais no código fonte considerando o escopo de aplicação de filtros de pré-processamento, o módulo de ambiente (***enviroment.py***) e o módulo de produção de vídeos utilizando os filtros do processamento (***filter_view.py***).

O módulo de ambiente é utilizado no fluxo de treinamento da rede convolucional para a obtenção do *frame* pré-processado, existe uma *thread* associada a um objeto que captura em tempo real o *frame* e aplica o processamento. No módulo foi alterado o *threshold* utilizado para considerar o novo modelo para treinamento e obtenção das recompensas para um posterior comparativo, tal comparativo foi impossibilitado devido aos pontos apresentados na primeira subseção deste capítulo.

Já o segundo módulo apresentava métodos para aplicar filtros em um vídeo predefinido. Como a principal finalidade do módulo era extrair informações sobre os *frames*, o mesmo foi expandido e parametrizado para considerar diferentes variações de técnicas, criar vídeos sobre cada uma das etapas e extrair métricas considerando todas as técnicas. foi feito, por meio do uso do módulo modificado, o comparativo entre as 5 técnicas selecionadas, os resultados são apresentados no próximo capítulo e estão persistidos no código fonte modificado relacionado a este trabalho.

4. Resultados

Foi proposto a realização do treinamento da agente utilizando diferentes algorítimos de limiarização durante a etapa de pre-processamento da imagem de entrada para Rede Neural. Conforme citado anteriormente, são *threshold triangle* (utilizado pelos autores do artigo), *threshold li*, *threshold niblack*, *threshold sauvola* e uma constante de limiarização C definida (após observar o resultado da limiarização para diferentes constante, foi definido a constante C = 25).

Conforme mostrados nas imagens abaixo, nenhum dos diferentes algoritmos de limiarização apresentou o resultado esperado durante o aprendizado do agente (incluído o algoritmo usado pelos autores do artigo).

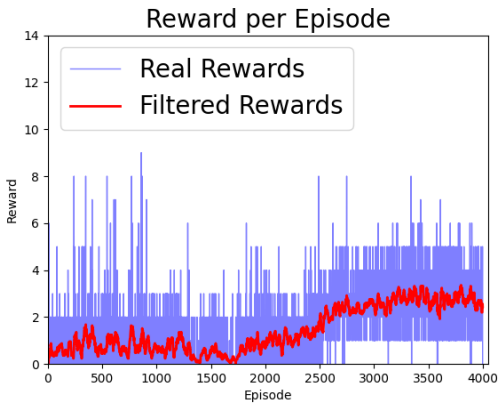

Figura 5. Gráfico de recompensa do *threshold niblack*

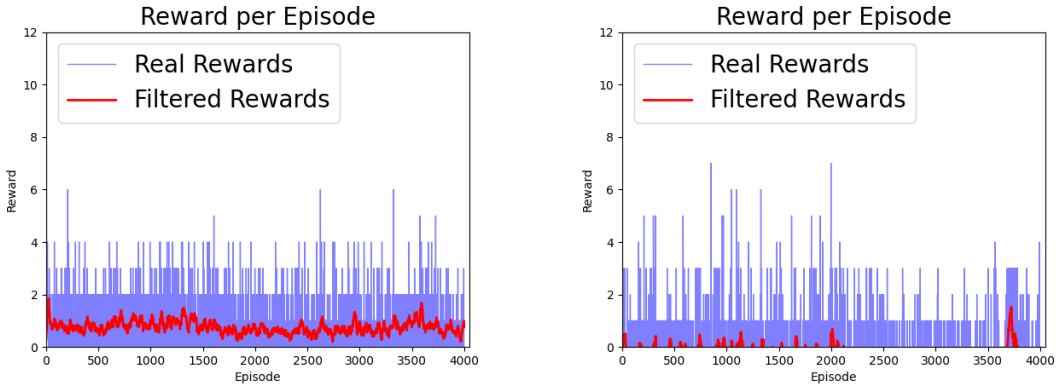

Figura 6. Gráfico de recompensa do *threshold triangle* e *threshold* constante

Conforme mostrado, a Figura 5 mostra o resultados obtidos utilizando o algorítimo *threshold niblack*, este foi o único que apresentou um pequena melhora após 2000 episódios, dito isto, o resultado ainda não se compara com o apresentado no artigo original. Nas demais figuras (Figuras 6 e 7), representados outros algorítimos de limiarização, não apresentaram melhora durante todo o processo de treinamento, permanecendo com uma média constante de recompensa.

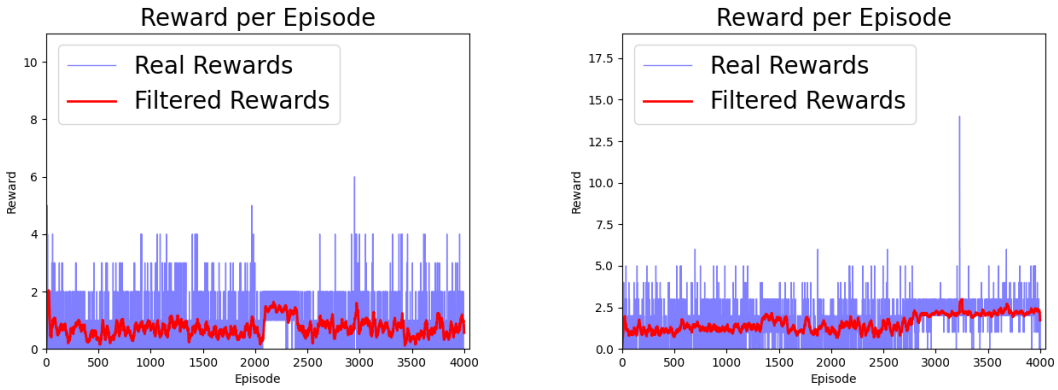

Figura 7. Gráfico de recompensa do *threshold li* e *threshold sauvola*

Devido a inconsistência do que foi apresentado no artigo e o que foi obtido como resultado a partir do código-fonte disponibilizado, foi decidido focar na etapa de pre-processamento, mais especificamente nos algorítimos de limiarização.

A ideia é analisar utilizando métricas qualitativas as imagens geradas pelos diferentes algorítimos de limiarização. Com base nos resultados obtidos por estas métricas pretendemos dizer os benefícios e desvantagens de utilizar um algorítimo ou outro, para um agente jogador de *Neon-Drive*.

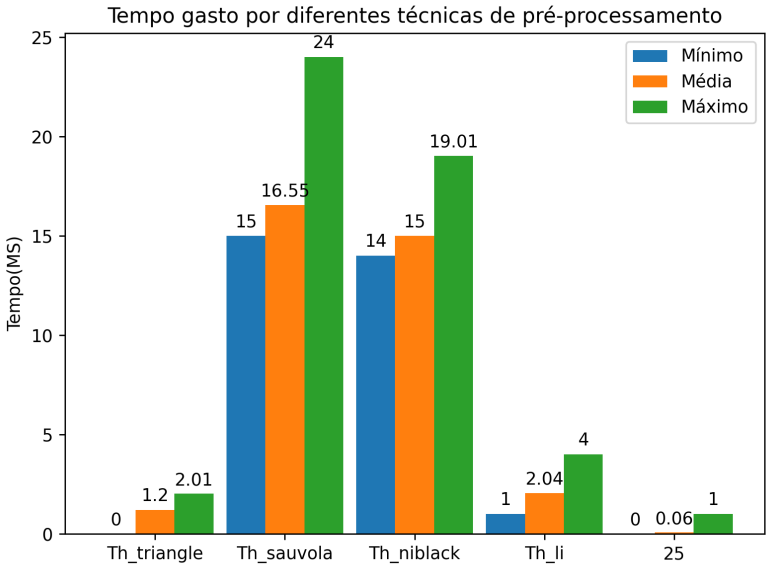

Figura 8. Tempo de execução dos algoritmos

A Figura 8 apresenta o tempo de execução em milissegundos (ms) para os diferentes algoritmos testados. Este gráfico tem como base verificar quanto tempo o agente de *Neon-Drive* deve esperar na etapa de limiarização, que impacta diretamente na latência de executar um movimento.

Dito isto, um algoritmo mais rápidos como *threshold triangle* e a constante de limiarização, tem preferência em ser utilizado. Porém, é importante notar que apesar de

mais lentos, os demais algoritmos são suficientes em um jogo como *Neon-Drive* onde poucas ações (3 a 5 no máximo) são necessárias no intervalo de um segundo.

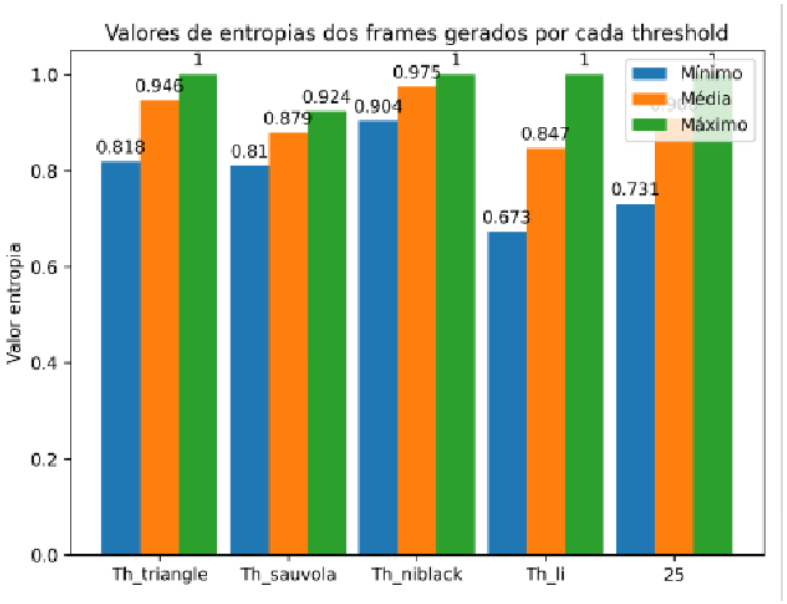

Figura 9. Entropia das imagens geradas pelos algoritmos

A Figura 9 apresenta a entropia das imagens geradas pelos diferentes algoritmos de limiarização. Com estas informações, obtemos a quantidade de informação em uma imagem gerada por determinado algorítimo. Acreditamos que uma imagem com pouca entropia (ou seja, como poucas informações), como as geradas pelo algoritmo *threshold li* (que apresentou a menor média de entropia), aceleram o aprendizado do agente, uma vez que este terá poucas informações para aprender.

A Figura 10 apresenta duas diferentes métricas para testar a similaridade das imagens geradas pelos diferentes algoritmos de limiarização: *Structural Similarity Index Measure* (SSIM) [Wang et al. 2004] e *Visual Information Fidelity* (VIF) [Sheikh et al. 2005]. A Figura 10 apresenta ainda, uma métrica usada para testar a qualidade das imagens geradas: *Universal Image Quality Index* (UQI) [Wang and Bovik 2002]. Todas essas métricas, foram obtidas observando a imagem original em escala de cinza (Uma das etapas de pre-processamento anteriores) com a imagem gerada após a execução dos algorítimos de limiarização.

Dito isso, essas métricas podemos dizer o quanto a imagem após o processos de limiarização representa da imagem original. Analisando os gráficos percebemos que os algoritmos *threshold sauvola* e *threshold nilblack* apresentam os melhores resultados de similaridade e qualidade em comparação com a imagem original em escala de cinzas.

Por fim, com as métricas de similaridade e qualidade em conjunto com a métrica de entropia, podemos dizer quanto das informações presentes na imagem gerada a partir do algorítimo de limiarização representam a imagem original. Dito isto, a escolha de um algoritmo de limiarização resulta *tradeoff* entre baixa entropia (e tempo de execução) com a alta similaridade e qualidade da imagem resultante.

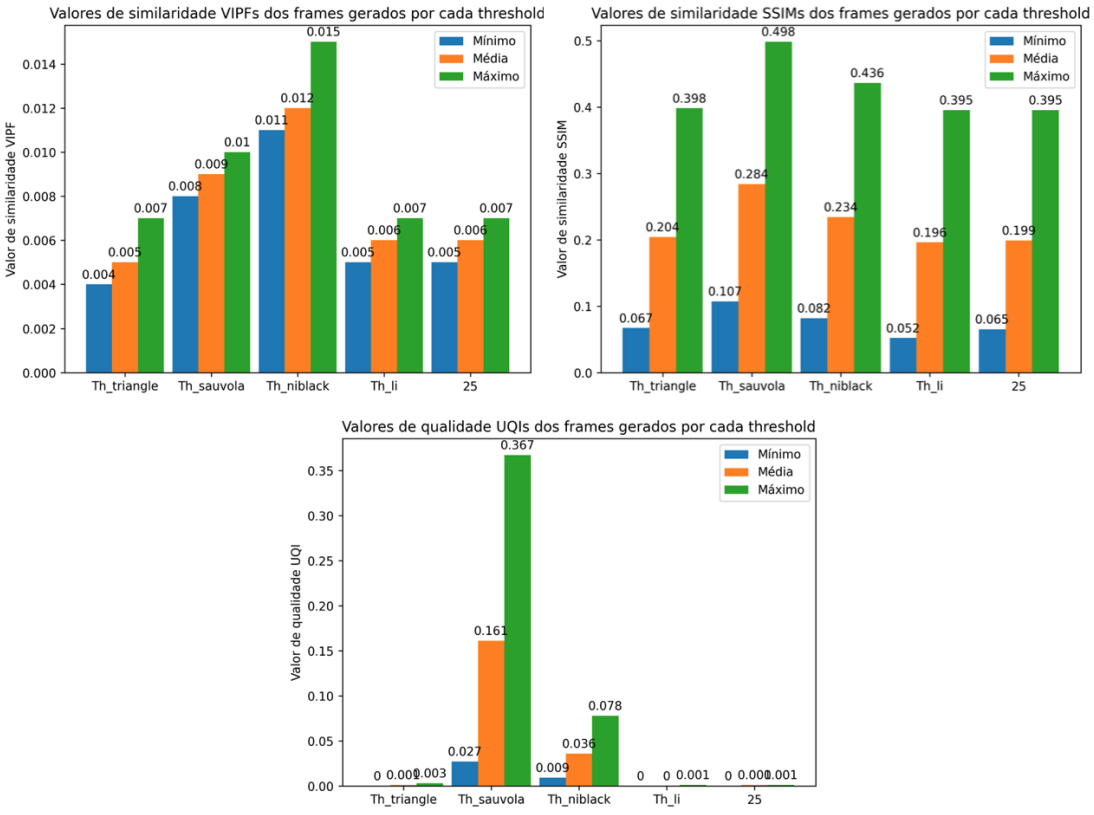

Figura 10. Métricas de Similaridade e Qualidade para os algoritmos

5. Conclusão

Podemos observar que devido ao erro no código do algoritmo de treinamento da *IA DQN* não foi possível obter um resultado minimamente satisfatório modificando etapas do pré-processamento de imagem, forçando mudanças de objetivos no projeto. Dessa forma o foco do trabalho se tornou analisar possíveis mudanças no processamento da imagem capturada em tempo real no jogo, utilizando cinco diferentes tipos de *thresholds*, desde o original usado pelo artigo até os mais diversos algoritmos encontrados, além do uso de constantes limiarização.

Ainda foi realizado alguns comparativos entre as limiarizações, utilizando cinco métricas distintas, como o tempo de execução, a entropia gerada, similaridade diferentes e qualidade dos *frames*. Com essas comparações, podemos concluir que não é possível definir o melhor algoritmo com base somente nas métricas, pois é possível analisar que nenhum algoritmo teve um excelente resultado em todos os testes. Para escolher o melhor *threshold* para o desafio proposto do artigo, é necessário escolher um que tenha uma baixa entropia e tempo de execução e uma alta similaridade e qualidade de imagem.

Concluindo esse estudo, é possível afirmar que existe grande inconsistência em que no artigo ele demonstra e explica a execução do projeto de uma maneira diferente do que é apresentado no código disponibilizado. Além disso, o código não treina a rede neural corretamente, mesmo que ele seja executado nas mesmas condições e ambientes do que foi realizados no projeto, que foi confirmado que algumas partes do pre-processamento não estavam funcionando. Por fim ainda é possível afirmar que o

artigo não explica, ou explica de uma maneira bem sucinta, algumas informações importantes que devem ser informadas para outras pessoas que queiram utiliza-las, como as configurações internas do jogo, explicação mais detalhada do código, explicação do porque escolher o otimizador *RMSprop*, entre outras.

Referências

Kich, V. A., de Jesus, J. C., Grando, R. B., Kolling, A. H., Heisler, G. V., and Guerra, R. d. S. (2022). Deep reinforcement learning using a low-dimensional observation filter for visual complex video game playing.

Sheikh, H., Bovik, A., and de Veciana, G. (2005). An information fidelity criterion for image quality assessment using natural scene statistics. *IEEE Transactions on Image Processing*, 14(12):2117–2128.

Wang, Z. and Bovik, A. (2002). A universal image quality index. *IEEE Signal Processing Letters*, 9(3):81–84.

Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.