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

Análise e comparação de desempenho e resultados

André Morais - Arthur Labaki - Vinicius Calixto - Vinnicius Pereira

Índice

- Relembrando
- Contribuições
- Objetivos
- Metodologia
- Resultados
- Pontos relevantes

Relembrando

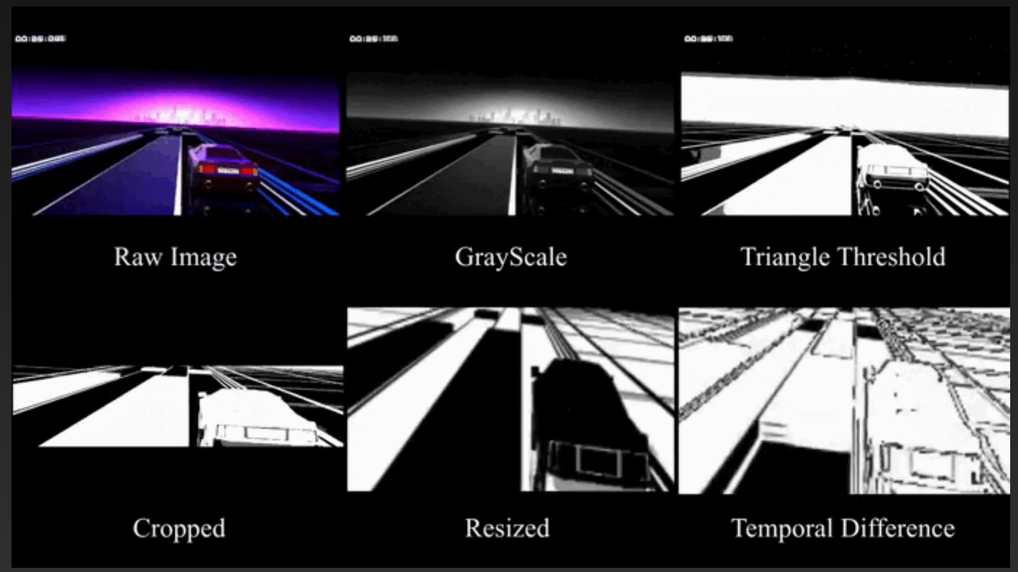
Relembrando

Neon Drive Reinforcement Learning

- Artigo selecionado: Deep Reinforcement Learning Using a Low-Dimensional Observation Filter for Visual Complex Video Game Playing
- Consiste em treinar um algoritmo de aprendizado de máquina (DRL) utilizando como objeto os frames capturados em tempo real do jogo Neon Drive
- Realiza um pré-processamento nos frames capturados para facilitar o aprendizado da IA
- Treina o algoritmo de acordo com uma função de recompensa, além de um teste de robustez

Relembrando
Etapa de pré-processamento

- Realizado em 6 etapas:
    1. Captura dos frames em BGR
    2. Conversão para escalas de cinza
    3. Limiarização (threshold) triangular
    4. Inversão de bits
    5. Corte e redimensionamento
    6. Diferença temporal



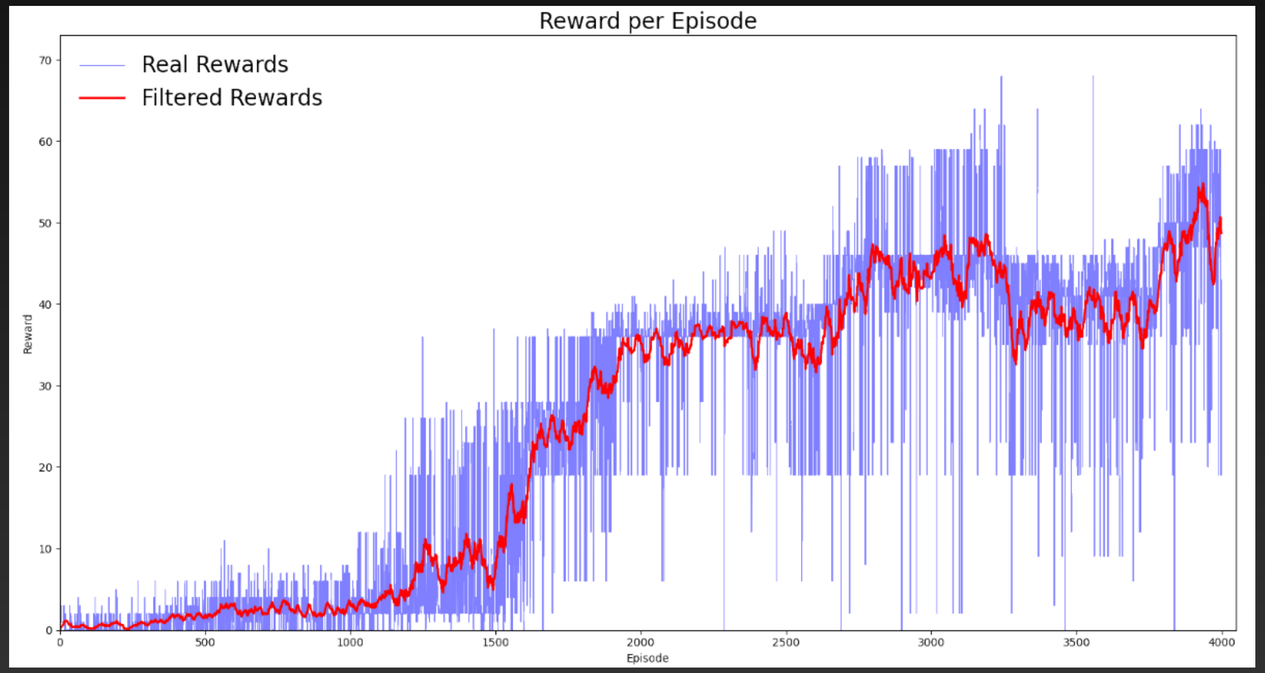
Relembrando

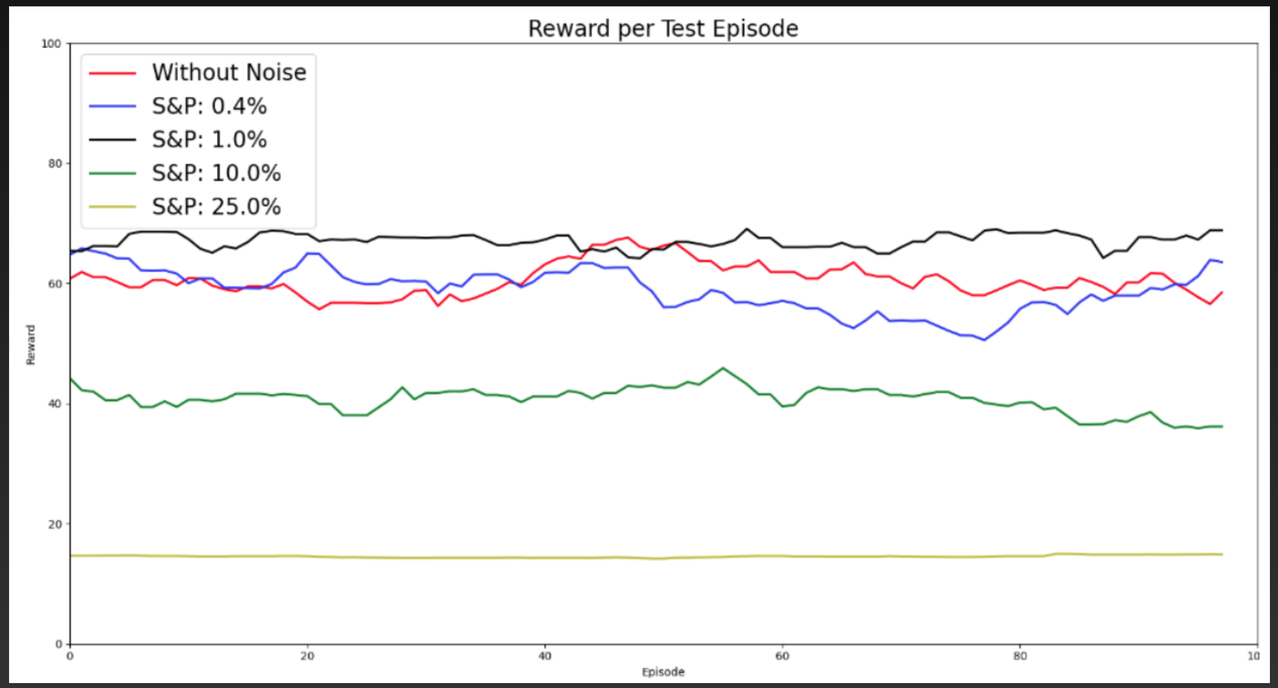
Resultados e teste de robustez

- Após o treinamento do algoritmo DRL, é gerado um gráfico de recompensa ruidoso, demonstrando a recompensa obtida em cada episódio, visando avaliar o uso do algoritmo
- Também é possível executar um teste de robustez para avaliar a técnica escolhida, inserindo o ruído sal com pimenta com diferentes porcentagens para as amostras

Relembrando

Resultados e teste de robustez





Relembrando
Conteúdo disponível

- Além do artigo disponível ainda existe o repositório do projeto, que contém o código-fonte, algumas informações básicas sobre o algoritmo e suas etapas de execução e algumas poucas instruções de como preparar o ambiente e iniciar o código.
- Também foi feito um vídeo resumido de todo o projeto.

- https://arxiv.org/pdf/2204.11370.pdf
- https://github.com/victorkich/Neon-Drive-DRL
- https://www.youtube.com/watch?v=bKdMBKu68QU&ab_channel=VersusAI

Contribuições

Contribuições

O que adicionamos ao projeto?

O trabalho propõe técnicas de pré-processamento de imagens para aumentar a eficiência das operações, como cortar frames e modificar a função de limiarização para obter melhores resultados. As modificações buscam melhorar a qualidade dos resultados e aplicar técnicas aprendidas em aulas de multimídia.

Objetivos

Objetivos

O que queria ser feito?

- Modificar etapas de pré-processamento e comparar com artigo.
- Adicionar novas etapas de pré-processamento e comparar com artigo.
- Utilizar outros tipos de ruído para teste de robustez.
- Modificação da ordem das operações de pré-processamento para comparar desempenho.
- Adição de novas etapas de pré-processamento, como compressão e filtros, para comparar resultados.
- Utilização de diferentes tipos de ruído para testar robustez dos resultados obtidos.

Objetivos

O que queremos com isso?

O trabalho busca otimizar o pré-processamento de imagens através da modificação e adição de etapas, bem como testar a robustez dos resultados com diferentes tipos de ruído.

Metodologia

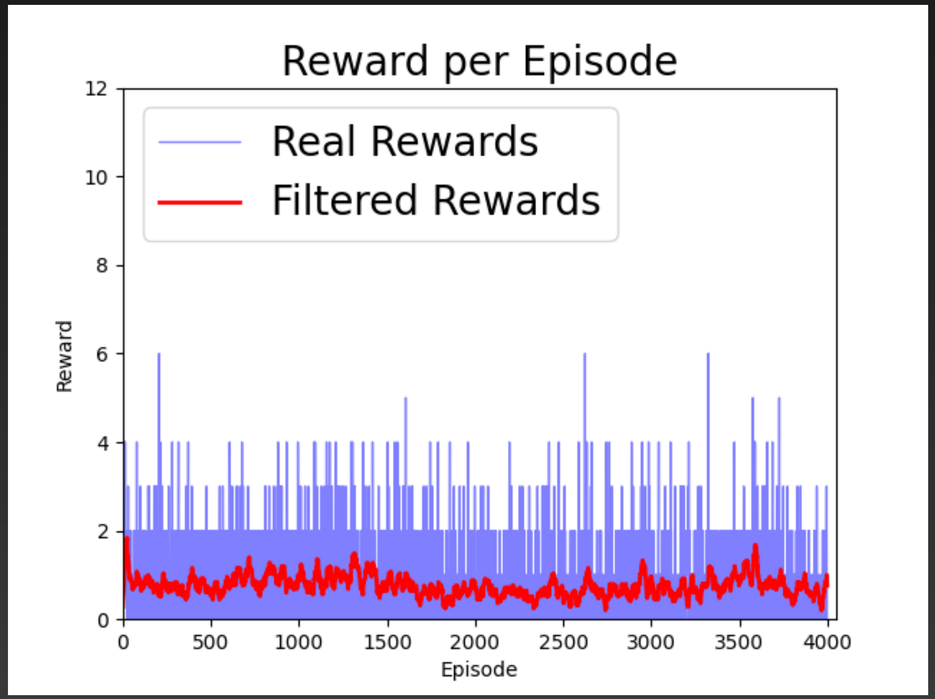
Metodologia

Etapas

- Treinamento da DQN
- Variação da ordem das etapas de pré-processamento
- Variação dos thresholds

Metodologia
Treinamento da DQN

- Problemas!
  - Treinamento não funcional
  - Vários testes
  - Modelo original não funciona
  - Falta de informação
  - Enfoque na etapa de pré-processamento



Metodologia
Mudanças do pré-processamento

- Mudanças no código
  - Fluxo de aprendizagem: environment.py
  - Aplicação de filtros em vídeos: filter_view.py

Metodologia

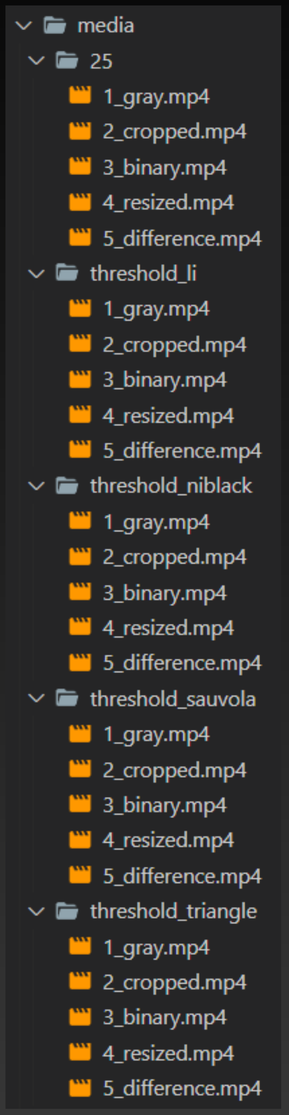
Mudanças do pré-processamento

- Mudanças: filter_view.py
  - Parametrização da extração de vídeos de filtros

```
# Creates 5 videos using preprocessing filters and return threshold metrics
def createVideosRetTimestamp(videoInputName, thresholdFunction, videoOutputDict, w = 1920, h = 1080):
```

  - Extração de métricas e comparativos automáticos

```
# Generate videos, extract metrics and save comparison graphics from thresholds list
def createPreProcessMetrics():

    thresholds = [threshold_triangle, threshold_sauvola, threshold_niblack, threshold_li, 25]
```



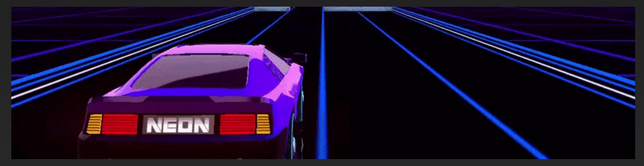
Metodologia

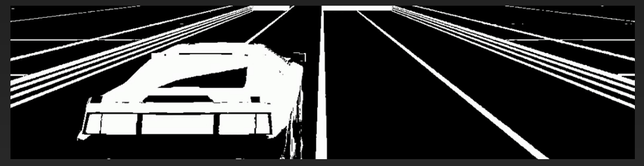
Variação da ordem das etapas de pré-processamento

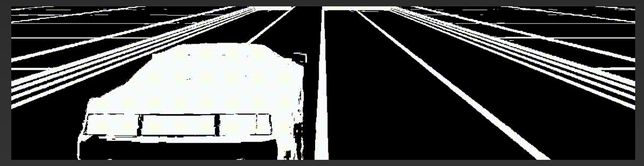
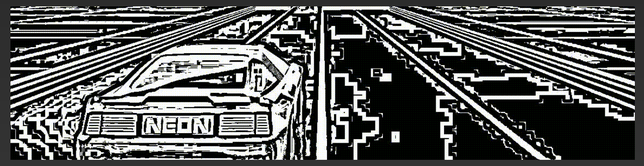
- Ordem de operações importa
  - Uma imagem processada por frame
    - 1000ms/60f ≅ 16msf
  - Corte posicionado após threshold
    - Threshold pode ser custoso
    - Melhor desempenho

Metodologia
Variação dos thresholds

- Seleção das técnicas
  - Testado mais de 10 técnicas
  - Seleção de 5 com uso de vídeos com filtros aplicados





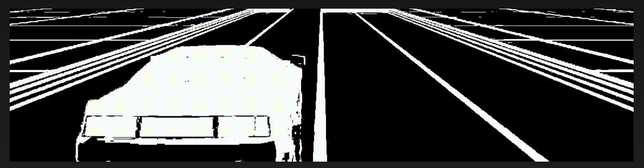
Metodologia
Variação dos thresholds

- Etapa do pré-processamento utilizada como auxiliar para a extração da imagem em bits.
- Comparação resulta em 1 (Verdadeiro) e 0 (Falso).
- Possível uso da constante ao invés do threshold fornece performance.

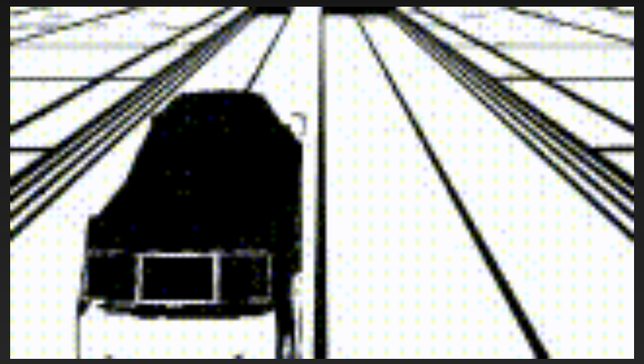
```
if type(thresholdFunction) != int:
    threshold_frame = thresholdFunction(cropped_bw_frame)
    binary_frame = cropped_bw_frame > threshold_frame

# int
else:
    binary_frame = cropped_bw_frame > thresholdFunction
```

Metodologia
Triangle Threshold

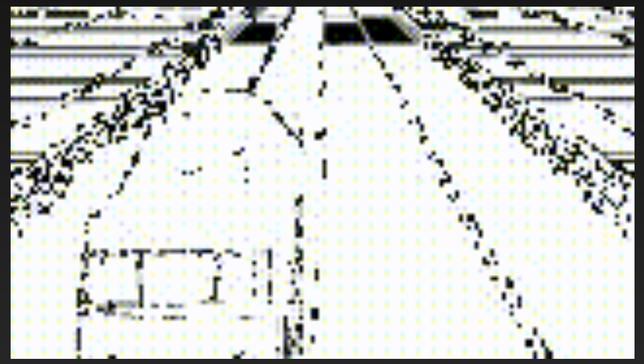
Threshold aplicado



Inversão de bits



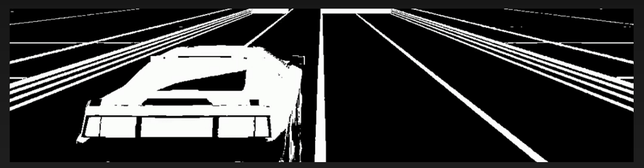
Diferença temporal



Threshold usado pelo artigo, que baseado no algoritmo do triângulo

Metodologia
Limiar constante

Threshold aplicado



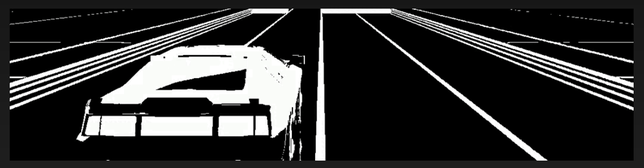
Inversão de bits

Diferença temporal

A constante é utilizada para obter a imagem binária, evitando processamento feito para calcular algum tipo de threshold

Metodologia
Li Threshold

Threshold aplicado



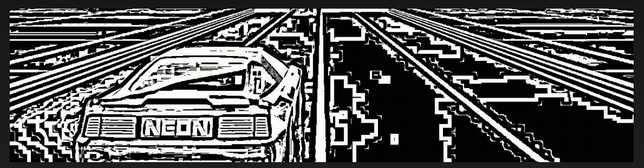
Inversão de bits

Diferença temporal

Calcula o valor do threshold utilizando o método iterativo de entropia cruzada mínima de Li

Metodologia
Niblack Threshold

Threshold aplicado



Inversão de bits

Diferença temporal

Originalmente projetado para reconhecimento de texto esse algoritmo aplica o threshold local Niblack em um array

Metodologia
Sauvola Threshold

Threshold aplicado

Inversão de bits

Diferença temporal

Sendo uma modificação do Niblack esse algoritmo aplica o threshold local Sauvola em um array

Metodologia

Métricas extraídas

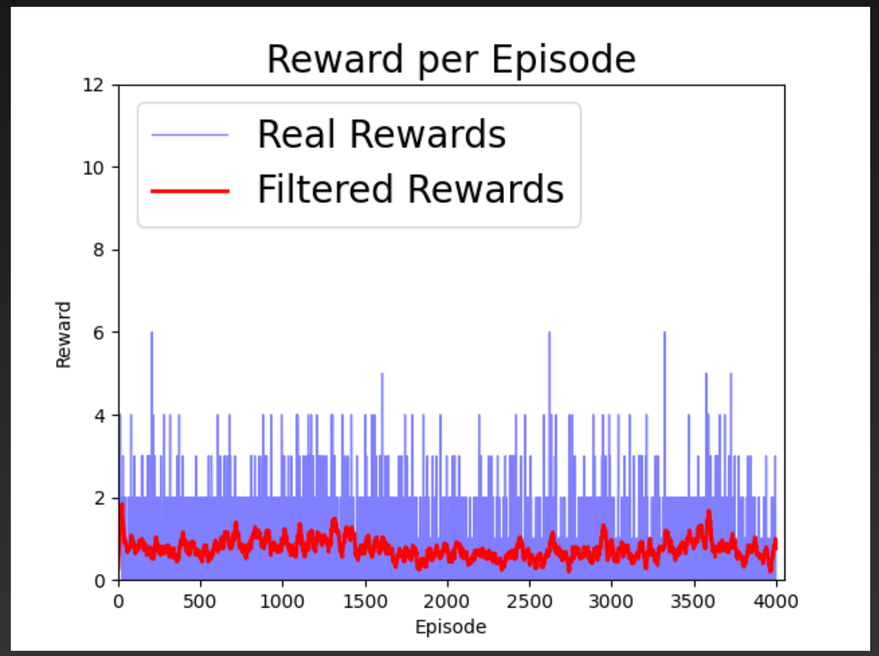
- Entropy
- Threshold timestamp
- Visual information fidelity (VIF)
- Structural similarity (SSIM)
- Universal image quality index (UQI)

Resultados

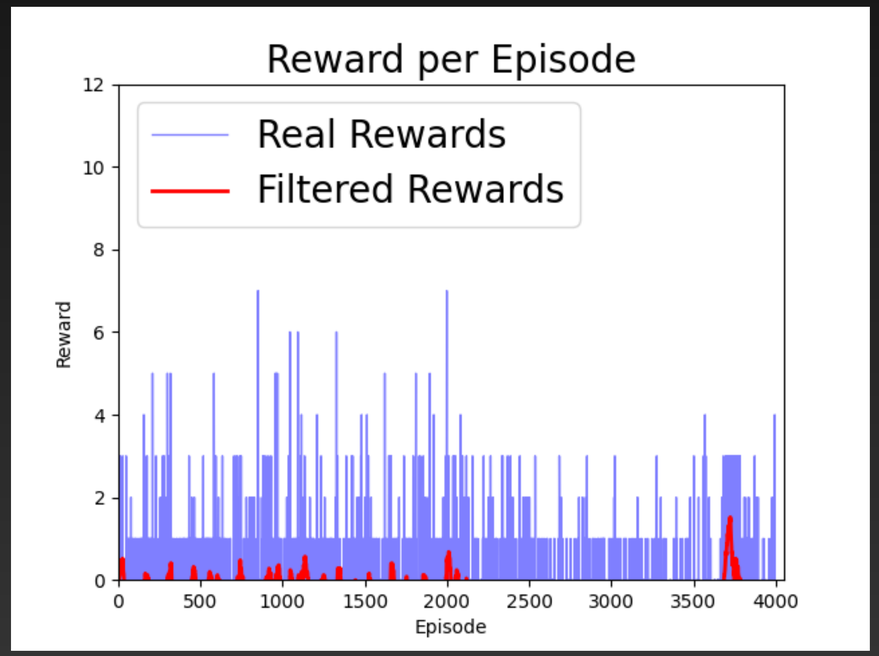
Resultados

Treinamento paras os diferentes Algoritmos de Limiarização

- Threshold Triangle



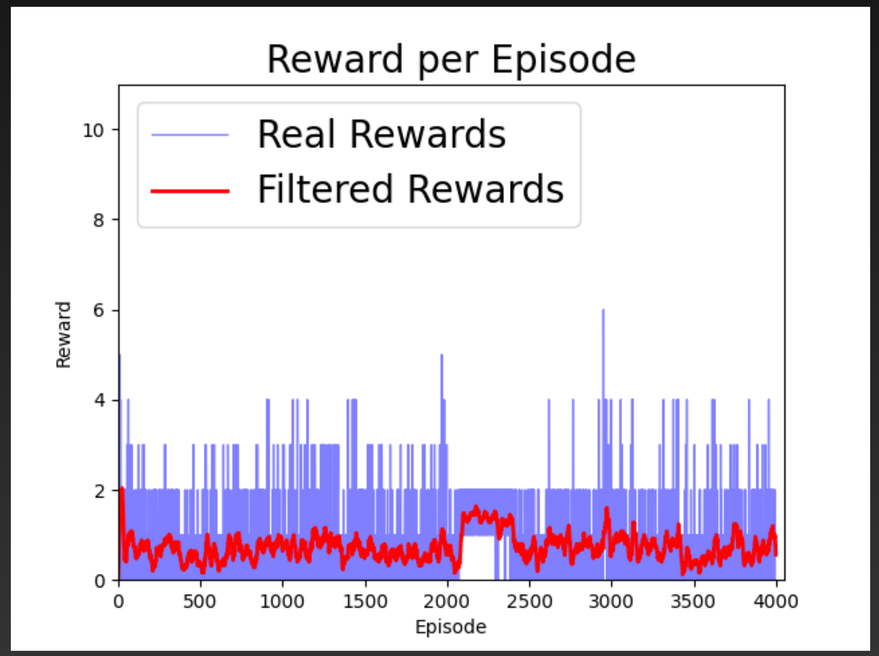
- Threshold Constant (C = 25)



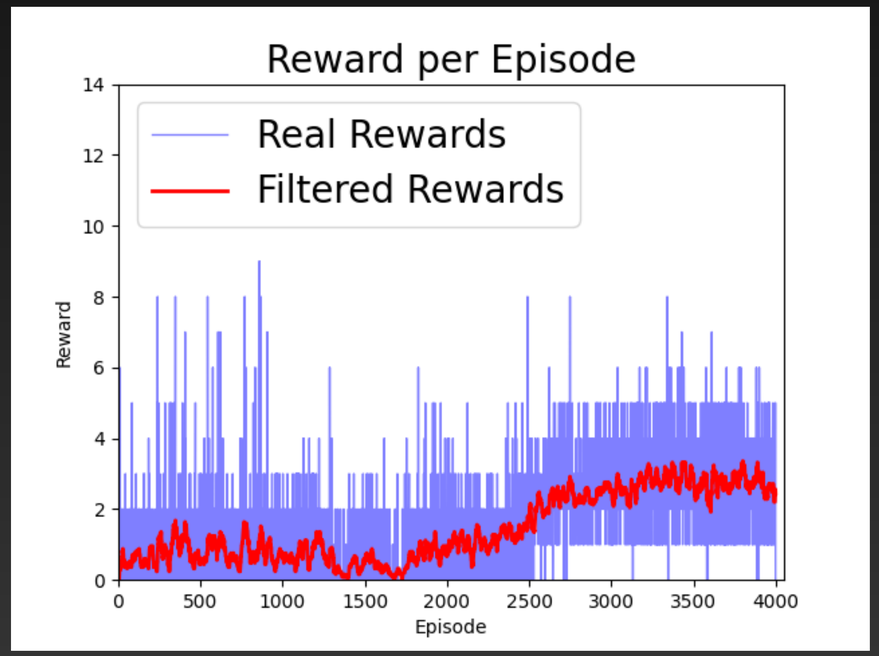
Resultados

Treinamento paras os diferentes Algoritmos de Limiarização

- Threshold Li



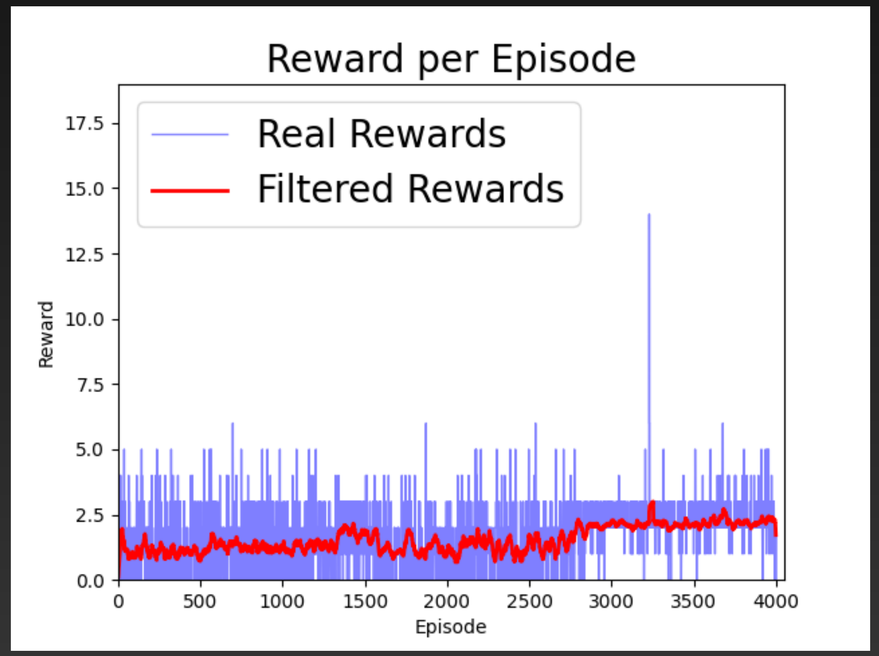
- Threshold Nilblack



Resultados

Treinamento paras os diferentes Algoritmos de Limiarização

- Threshold Sauvola

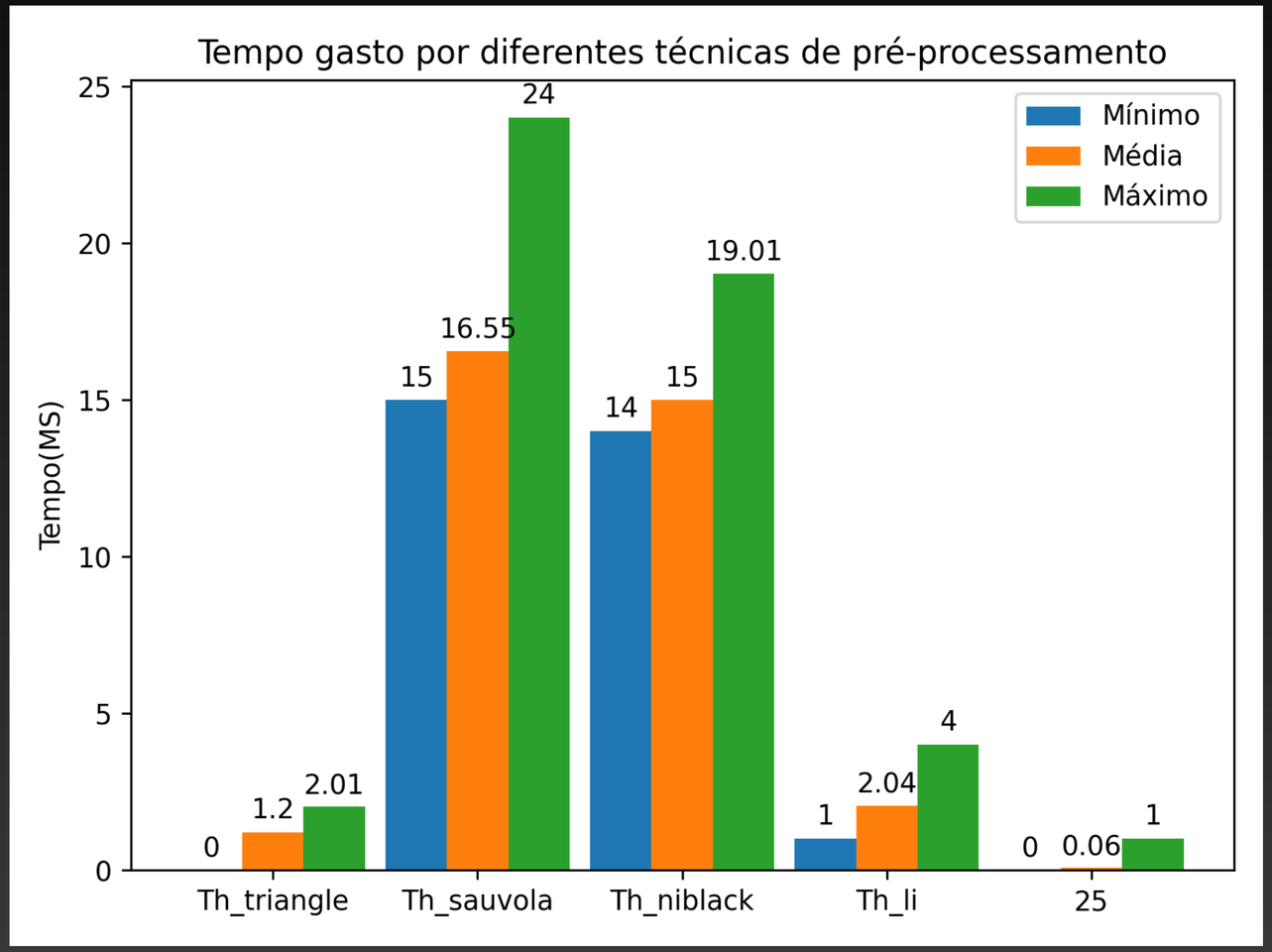


Resultados

- Foco nas análise das imagens geradas pelos diferentes algoritmos de Limiarização.
- Obter métricas para analizar estas imagens.
- Identificar os benefícios e desvantagens de utilizar um algoritmo ou outro.

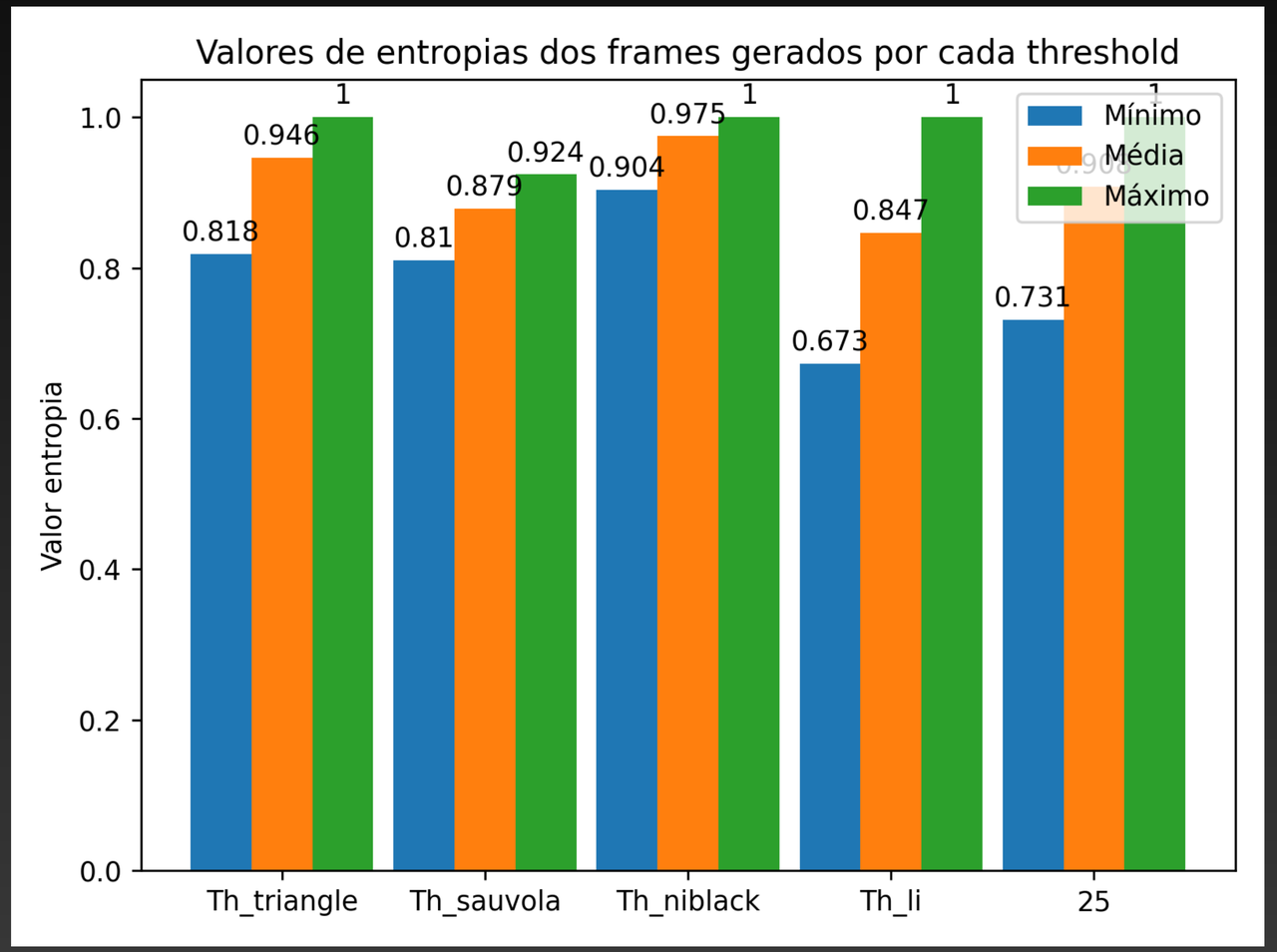
Resultados

- Métrica: Tempo de Execução



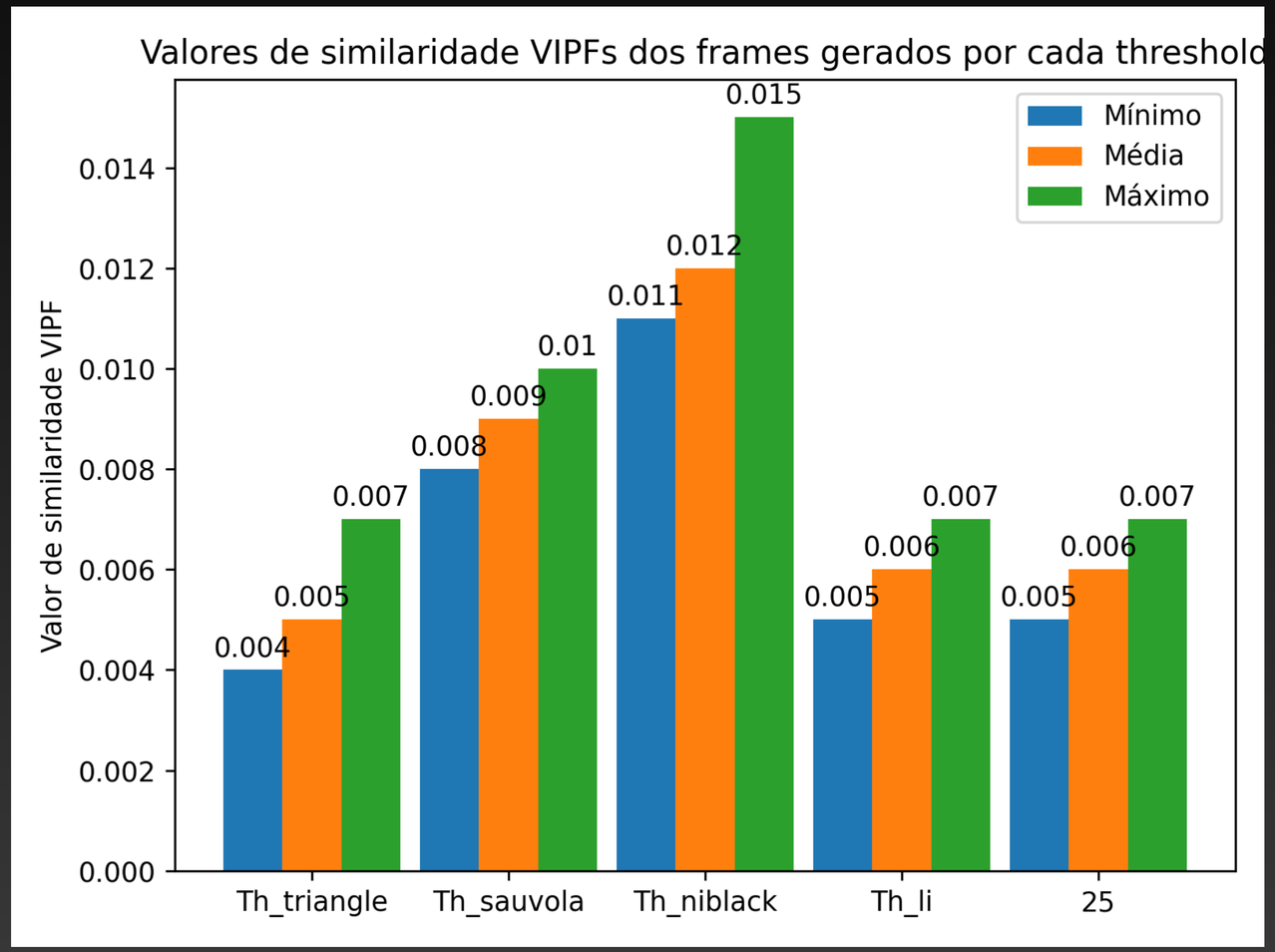
Resultados

- Métrica: Entropia



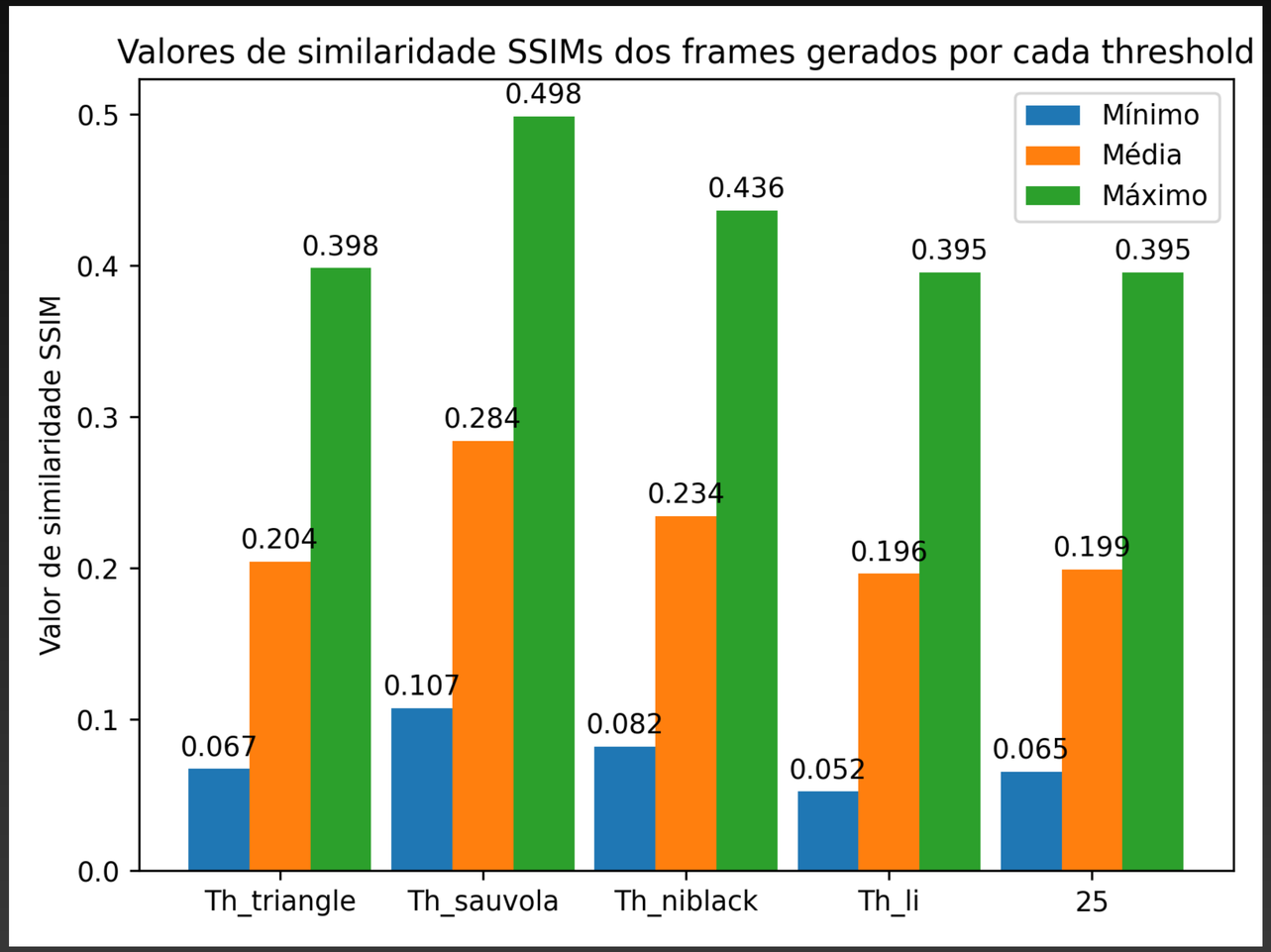
Resultados

- Métrica: Similaridade



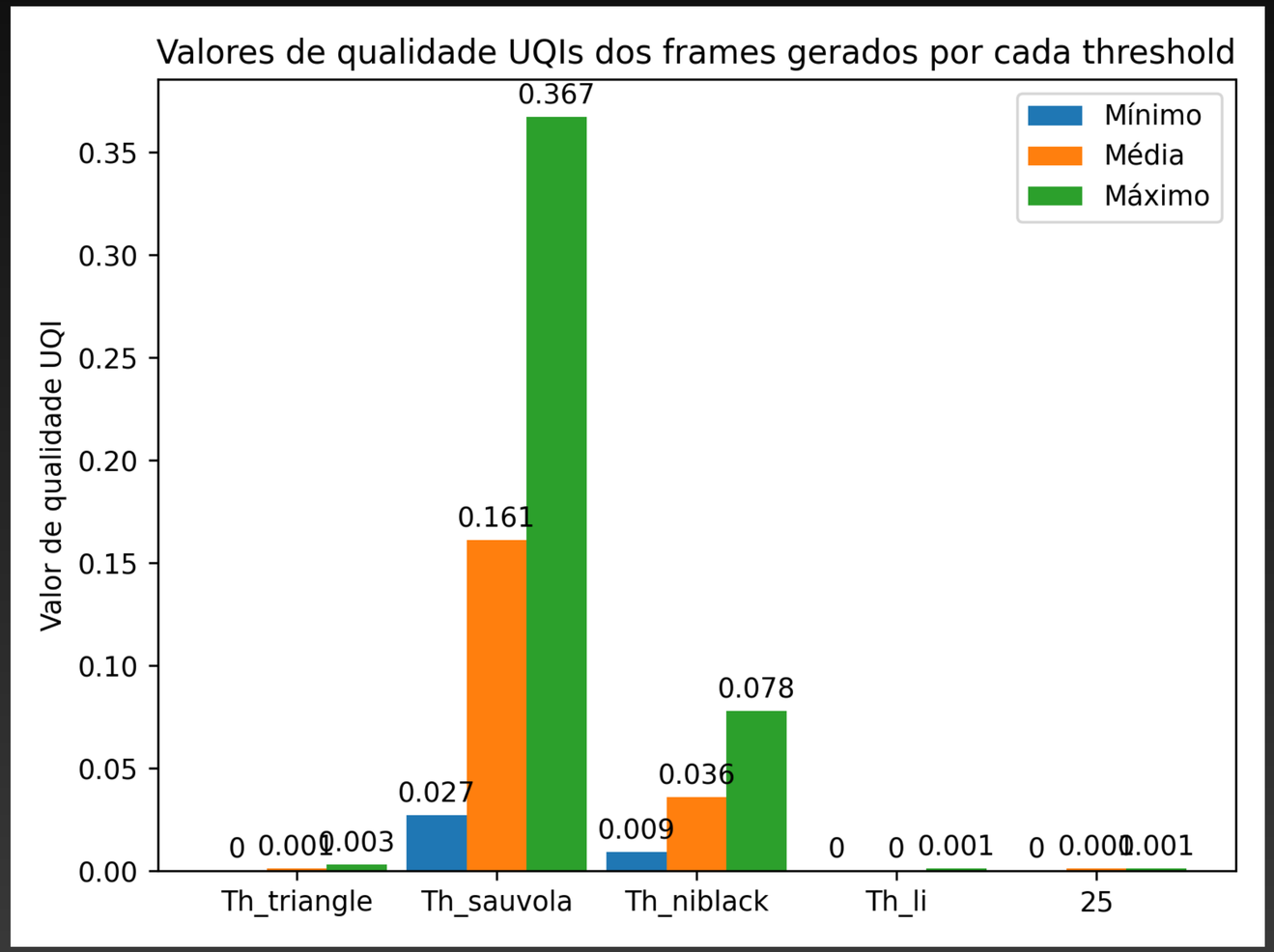
Resultados

- Métrica: Similaridade



Resultados

- Métrica: Qualidade



Pontos relevantes

Pontos relevantes

Conclusão

- Independente do tratamento da imagem utilizado, o resultado do treinamento não foi satisfatório
- Cada threshold tem uma característica especifica para determinada função
- O melhor threshold para o nosso problema seria um com baixa entropia e tempo de execução e uma alta similaridade e qualidade de imagem.

Pontos relevantes

Opiniões

- Código disponibilizado não treina a IA de maneira correta, mesmo com um ambiente idêntico e sem alterações.
- O artigo está inconsistente com o código e vídeo, explicando etapas de pré-processamento em ordem diferente.
- O artigo não informa alguns pontos chaves para sua replicação, como configuração do jogo, detalhamento do código, motivos para o uso de certas ferramentas, entre outros.