UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MACELA ZOVKA DE MORAES LEMOS

CONSTRUÇÃO DE APLICAÇÃO WEB PARA HOSPEDAGEM DE ALGORITMO DE AVALIAÇÃO AUTOMÁTICA DA QUALIDADE DE IMAGENS UTILIZANDO O DOCKER

Recife
2024

MACELA ZOVKA DE MORAES LEMOS

CONSTRUÇÃO DE APLICAÇÃO WEB PARA HOSPEDAGEM DE ALGORITMO DE AVALIAÇÃO AUTOMÁTICA DA QUALIDADE DE IMAGENS UTILIZANDO O DOCKER

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. **Área de Concentração:** Inteligência Computacional.

Orientador: Prof. Dr. Carlos Alexandre Barros de Mello

Recife

2024

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Lemos, Macela Zovka de Moraes.
Construção de aplicação web para hospedagem de algoritmo de avaliação automática da qualidade de imagens utilizando o Docker / Macela Zovka de Moraes Lemos. - Recife, 2024.
74f.: il.

Dissertação (Mestrado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciências da Computação, 2024.
Orientação: Carlos Alexandre Barros de Mello.

1. Processamento Digital de Imagens; 2. Avaliação da Qualidade de Imagens; 3. Docker; 4. Aplicação Web. I. Mello, Carlos Alexandre Barros de. II. Título.

UFPE-Biblioteca Central

Macela Zovka de Moraes Lemos

"CONSTRUÇÃO DE APLICAÇÃO WEB PARA HOSPEDAGEM DE ALGORITMO DE AVALIAÇÃO AUTOMÁTICA DA QUALIDADE DE IMAGENS UTILIZANDO O DOCKER"

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Inteligência Computacional.

Aprovado em: 31/07/2024.

BANCA EXAMINADORA

_____

Prof. Dr. Alexandre Cabral Mota
Centro de Informática / UFPE

_____

Profa. Dra. Mylène Christine Queiroz Farias
Departamento de Engenharia Elétrica / UnB

_____

Prof. Dr. Carlos Alexandre Barros de Mello
Centro de Informática / UFPE
(**orientador**)

Dedico este trabalho à minha família, minha mãe, minhas irmãs e ao meu pai. Até aqui nos ajudou o Senhor.

AGRADECIMENTOS

Agradeço ao meu Professor Orientador Carlos Alexandre Barros de Mello pela oportunidade, instrução e todo o apoio depositado. Agradeço também e aproveito para expor minha admiração e respeito aos profissionais competentes aqui formados e que compõem o Centro de Informática da UFPE.

Agradeço ao Paulo Guarda e à equipe da Motorola pelo apoio e provimento de um ambiente favorável ao meu crescimento profissional e acadêmico; aos meus colegas de trabalho, em especial ao Ruan, Pedro, Daniel e Serafim, pelo suporte em momentos de dificuldade fundamental para conclusão deste trabalho; e a toda a equipe do Projeto CIn/Motorola, com quem tive o prazer de conviver durante este período.

RESUMO

A avaliação automática da qualidade de imagens permite a atribuição de notas a imagens digitais de forma objetiva. Há diversos algoritmos desenvolvidos para executar essa avaliação, a qual pode ser feita tendo ou não alguma imagem de referência para fins de comparação. Recentemente, redes neurais convolucionais têm sido usadas para uma previsão acurada da qualidade de imagens sem referência. Neste sentido, a criação de uma ferramenta *web*, na qual um algoritmo avaliador de imagens está embarcado permite que usuários façam uso de métodos de predição de qualidade de imagens.

Este trabalho detalha a construção de uma ferramenta *web* denominada "IQA Tool", designada a hospedar algoritmos de avaliação automática de imagens sem referência para atribuição de notas às imagens submetidas. Ainda, trata da ambientação dessa ferramenta à uma máquina de sistema operacional Windows 11 por meio de uma popular plataforma de criação de contêineres virtuais, o Docker, que é gratuito para uso pessoal. Neste estudo são descritos os conceitos fundamentais para o entendimento dos componentes envolvidos, como: funcionamento do Docker, implementação da IQA Tool na máquina hospedeira e avaliação de imagens utilizando diferentes metodologias de avaliação automática de imagens sem referência.

Palavras-chave: Processamento Digital de Imagens, Avaliação da Qualidade de Imagens, Docker, Aplicação *Web*.

ABSTRACT

The automatic evaluation of image quality allows for the objective assignment of scores to digital images. Various algorithms have been developed to perform this evaluation, which can be done with or without a reference image for comparison purposes. Recently, convolutional neural networks have been used for accurate prediction of image quality without a reference. In this context, the creation of a web tool, where an image evaluation algorithm is embedded, allows users to utilize image quality prediction methods.

This work details the construction of a web tool called "IQA Tool", designed to host no-reference automatic image evaluation algorithms for scoring submitted images. Additionally, it addresses the setup of this tool on a Windows 11 operating system machine using a popular virtual container creation platform, Docker, which is free for personal use. This study describes the fundamental concepts for understanding the components involved, such as the operation of Docker, the implementation of the IQA Tool on the host machine, and the evaluation of images using different no-reference automatic image evaluation methodologies.

Keywords: Digital Image Processing, Image Quality Assessment, Docker, Web Application.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BLIINDS	*Notador de Integridade de Imagem Cega usando Estatísticas DCT*
BRISQUE	*Avaliador de Qualidade de Imagem Espacial sem Referência*
CBIQ	*Qualidade de Imagem Baseada em Código Livro*
CNN	*Redes Neurais Convolucionais*
DIIVINE	*Avaliação da Veracidade e Integridade da Imagem Baseada na Identificação de Distorções*
DIQA	*Preditor de Avaliação de Qualidade de Imagem Profunda*
FR	*Full Reference*
FR-IQA	*Full-Reference Image Quality Assessment*
GAP	*Global Average Pooling*
GB	*Gaussian Blurr*
GPU	*Unidade de Processamento Gráfico*
HRIQ	*High Resolution Image Quality*
IQA	*Image Quality Assessment*
LBIQ	*Qualidade de Imagem Baseada em Aprendizado*
NR	*No-Reference*
NR-IQA	*No-Reference Image Quality Assessment*
NSS	*Natural Scene Statistics*
RELU	*Unidade linear retificada*
RR	*Reduced Reference*
VMs	*Máquinas Virtuais*
WSL 2	*Windows Subsystem for Linux 2*

LISTA DE SÍMBOLOS

$\sum$	Somatório
θ	Teta
σ	Sigma
μ	Mi
α	Alfa

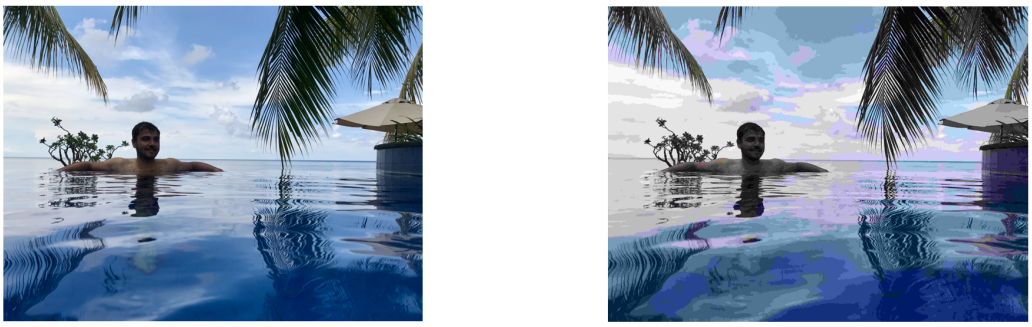
SUMÁRIO

1 INTRODUÇÃO

Devido aos avanços recentes na tecnologia da informação, conforme aponta Jorgenson e Vu (2016), e ao uso generalizado de *smartphones*, que permitem a rápida aquisição de informações e comunicação entre pessoas, imagens digitais são onipresentes na sociedade contemporânea. A estimativa exposta em Redi et al. (2015) é de que bilhões de imagens são criadas, compartilhadas e, finalmente, apreciadas pelos usuários todos os dias. No entanto, de acordo com Yu et al. (2019a) e Santokhi (2017), as imagens digitais são sensíveis a uma grande variedade de distorções durante o mecanismo de entrega pelo qual passam.

Conforme descrito por Santokhi (2017), esse mecanismo começa com a aquisição de uma imagem por uma câmera digital, onde a intensidade e a cor da luz são capturadas por um sensor que as transforma em bits, que são então armazenados em forma comprimida na memória interna da câmera. A imagem pode então ser enviada por um canal de transmissão para ser recebida por outro usuário. O usuário final *consome* a imagem em uma ampla variedade de dispositivos. Durante qualquer uma das etapas desse mecanismo de entrega, a imagem está sujeita a vários tipos de distorções, como ruído ou borrão. A Figura 1 traz o exemplo de uma possível distorção de imagem causada por um mecanismo de entrega de imagem e a Figura 2 ilustra a distorção causada pela compressão JPEG.

Figura 1 – Exemplo de uma possível distorção de imagem causada por um mecanismo de entrega de imagem.



(a) Foto original. (b) Foto distorcida após entrega.

Fonte: Extraído de Santokhi (2017).

O texto de Santokhi (2017) relaciona como essas possíveis distorções podem prejudicar a qualidade perceptual de uma imagem e causar insatisfação ao usuário que, por sua vez, pode optar por mudar de provedor de conteúdo ou serviço, ou comprar outra câmera digital, ou dispositivo de consumo. Sob essa motivação, os provedores de serviços e dispositivos buscam otimizar o fluxo de processamento e obter resultados perceptualmente bons. Para isso, é necessário estimar a qualidade perceptual da imagem.

Figura 2 – Uma imagem de referência de alta qualidade e sua versão comprimida no formato JPEG, com alta taxa de perda.

(a) Foto original. (b) Foto degradada pela compressão.

Fonte: Extraído de Yu et al. (2019b).

A avaliação da qualidade da imagem (*Image Quality Assessment* (IQA)) é voltada à medição da degradação das imagens de forma compatível com a percepção humana, já que os humanos são os julgadores finais da qualidade entregue de uma imagem. Para alcançar isso, é necessária uma função objetiva que reflita a percepção humana da degradação, conforme aponta Mohammadi, Ebrahimi-Moghadam e Shirani (2014).

Alguns artigos também apontam a relevância de algoritmos de avaliação da qualidade de imagem e seus significativos impactos nas indústrias de *smartphones* (FANG et al., 2020) e também em aplicações práticas das áreas médicas (CHOW; PARAMESRAN, 2016), (BOBY; SHARMIN, 2021), automotivas (HACHEM et al., 2021), (HüTTEN et al., 2024), de segurança (KLIMA et al., 2007), (LI et al., 2018), (GALBALLY; MARCEL; FIERREZ, 2014), midiáticas (JAIN; TANEJA; TANEJA, 2024), (VO; TRAN; LE, 2017) e (FONG et al., 2019), entre outras. O desenvolvimento desses algoritmos é impulsionado pela necessidade de automatizar processos manuais e aumentar a eficiência operacional associada às tarefas em que são empregados.

De forma geral, a função objetiva do IQA é um modelo matemático que, com base na análise dos valores dos pixels da imagem, consegue prever a qualidade perceptual de uma imagem. O trabalho desenvolvido pelo Santokhi (2017) expõe que essas funções podem ser classificadas conforme a disponibilidade de uma imagem de referência, como é chamada a imagem original, sem nenhuma distorção aplicada a ela.

Os métodos com referência completa, *Full Reference* (FR), realizam a comparação entre duas imagens. Uma imagem natural é fornecida como referência, sendo utilizada como comparativo para estabelecer a qualidade da imagem degradada em questão. Conforme exposto por SILVA (2022), os métodos FR são métodos bem situados na literatura atual e, geralmente, são rápidos e eficientes.

O trabalho desenvolvido pelo Varga (2019), por exemplo, fornece uma avaliação abran-

gente de 32 algoritmos FR IQA de ponta, utilizando o banco de dados MDID publicado por Sun, Zhou e Liao (2017), que contém imagens de referência e suas imagens derivadas, distorcidas utilizando tipos e níveis aleatórios de distorções. Neste estudo, foram consideradas distorções como ruído Gaussiano, desfoque Gaussiano, alteração de contraste, ruído JPEG e ruído JPEG2000. Esses métodos, no entanto, restringem-se ao cenário onde a imagem original livre de distorções está disponível.

Na abordagem sem referência, *No-Reference Image Quality Assessment* (NR-IQA), os métodos empregados aprendem a prever o julgamento humano da qualidade de imagem a partir de bancos de dados de imagens avaliadas por humanos. Como os métodos *Avaliador de Qualidade de Imagem Espacial sem Referência* (BRISQUE) (MITTAL; MOORTHY; BOVIK, 2012), DIIVINE (MOORTHY; BOVIK, 2012), BLIINDS (SAAD; BOVIK; CHARRIER, 2012), CBIQ (YE; DOERMANN, 2012), LBIQ (TANG; JOSHI; KAPOOR, 2011) e do *Preditor de Avaliação de Qualidade de Imagem Profunda* (DIQA) (KIM; NGUYEN; LEE, 2019). Este último usa *Redes Neurais Convolucionais* (CNN) na avaliação da qualidade de imagens sem referência e adota uma estrutura de treinamento em duas etapas para obter uma alta precisão na avaliação das imagens.

Em contrapartida, o trabalho apresentado por Mittal, Soundararajan e Bovik (2013) desenvolve o NIQE, o qual é um modelo NR baseado em *Natural Scene Statistics* (NSS), que não requer exposição prévia a imagens distorcidas ou qualquer treinamento com base em notas atribuídas por humanos.

Em intermédio, está a avaliação *Reduced Reference* (RR) onde apenas algumas características são extraídas das imagens de referência e comparadas às características presentes na imagem distorcida a ser analisada. Por meio da comparação de características específicas, a pontuação da qualidade da imagem é atribuída. No entanto, a escolha acurada de quais elementos devem ser avaliados nas imagens é fundamental para a eficácia dos métodos RR.

De acordo com Santokhi (2017), os métodos NR são de maior interesse para as áreas de pesquisa por preverem a qualidade perceptual de uma imagem, mesmo que nenhuma informação da imagem de referência esteja disponível. Isso se confirma com um exemplo prático de aplicabilidade dos métodos NR nos testes de produtos, mais especificamente câmeras de *smartphones*.

É neste contexto que se origina este trabalho. Uma versão semelhante à ferramenta aqui desenvolvida é atualmente aplicada no ambiente de testes de produtos da Motorola, hospedado no Centro de Informática da Universidade Federal de Pernambuco, para auxiliar na identificação de problemas de câmeras. Notas baixas obtidas pela ferramenta devido à presença de borramentos, ruídos ou áreas escuras nas imagens, ajudam a identificar dispositivos com defeitos e seus pontos de melhoria. A Figura 3 ilustra uma imagem com borramento extraída do banco de dados público, o HRIQ (HUANG QIANG WAN, 2024), obtida por um dispositivo móvel, ilustrando um problema que pode acontecer na

imagem, tornando-a de baixa qualidade. Nesse caso, o problema foi gerado na aquisição da foto, mas reflete o que propomos avaliar aqui.

Figura 3 – Fotografia com borramento obtida por um dispositivo celular. Imagem sem referência extraída do banco de dados HRIQ (HUANG QIANG WAN, 2024).

Fonte: Banco de dados HRIQ (HUANG QIANG WAN, 2024)

O desenvolvimento de uma ferramenta para avaliação automática da qualidade de imagens com interface *web* permite que testadores interajam de forma prática com o sistema. Embora exista um esforço inicial para configurar a ferramenta, após sua instalação, o custo operacional é mínimo. Por exemplo, não é necessário modificar nenhum código ou diretório para avaliar novos conjuntos de imagens.

Além disso, a utilização de ferramentas de conteinerização para hospedagem e desenvolvimento de aplicações está alinhada com as práticas atuais do grupo de trabalho CIn/Motorola. Estas práticas têm por objetivo facilitar as instalações, assegurando a portabilidade e o funcionamento da ferramenta em qualquer ambiente hospedeiro, seja em computadores pessoais com sistemas operacional Microsoft Windows ou Linux, ou em máquinas virtuais empresariais para uso coletivo, como é o caso desenvolvido no grupo citado.

Com base no exposto, o desafio que este trabalho enfrenta é a implementação de uma ferramenta *web* com interface amigável, capaz de hospedar métodos de avaliação de qualidade de imagem sem referência (NR-IQA). A ferramenta ajudará na identificação de imagens de baixa qualidade, podendo indicar algum defeito de câmera a ser posteriormente confirmado. Diferentes métodos de NR-IQA podem ser utilizados, por meio da modificação do código-fonte da aplicação desenvolvida. Uma vez configurada, o usuário precisa apenas fornecer as imagens a serem avaliadas, recebendo como retorno as pontuações com as

avaliações correspondentes.

A estrutura da ferramenta proposta suporta qualquer modelo NR-IQA escrito em Python para realizar a avaliação das imagens. Para ilustrar as etapas necessárias para modificar o código-fonte da ferramenta, este trabalho apresenta dois cenários: no primeiro, o sistema é configurado para hospedar o algoritmo DIQA, e no segundo, para o algoritmo BRISQUE.

O algoritmo DIQA, proposto por Kim, Nguyen e Lee (2019), utiliza uma abordagem baseada em redes neurais convolucionais (CNNs) e é treinado em duas etapas. Na primeira, a CNN aprende a prever um mapa de erro objetivo; na segunda, o modelo aprende a prever a pontuação subjetiva de qualidade. A ferramenta desenvolvida neste trabalho não inclui o treinamento da CNN, mas embarca uma rede já treinada utilizando este método. O DIQA foi escolhido devido ao seu uso anterior pelo grupo de pesquisa de avaliação automática da qualidade de imagens no teste de produtos, contexto no qual a autora está inserida.

O segundo modelo adotado, o BRISQUE, proposto por Mittal, Moorthy e Bovik (2012), segue uma abordagem distinta, baseada em características estatísticas extraídas de mapas de intensidade dos pixels para prever a qualidade da imagem. O BRISQUE combina medidas de distorção espacial natural e regressão para fornecer uma estimativa da qualidade sem referência. Diferente do DIQA, o BRISQUE não utiliza redes neurais convolucionais, sendo um código mais leve e já disponível no repositório oficial de pacotes Python (<https://pypi.org/project/brisque/>). Esses pontos levaram à escolha desse segundo método.

1.1 OBJETIVOS GERAIS

O objetivo geral deste trabalho é apresentar o desenvolvimento de uma interface *web*, com um algoritmo NR-IQA embarcado. Essa interface é de fácil instalação e uso, permitindo que os usuários realizem a avaliação automática da qualidade de imagens diretamente em seus computadores pessoais, com pequenas adaptações, dependendo do algoritmo em uso. O desenvolvimento utiliza exclusivamente recursos gratuitos e acessíveis ao público no momento da criação deste trabalho.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho são:

- Criar uma interface *web* intuitiva: associar à ferramenta uma interface *web* de fácil uso, permitindo uma interação amigável e eficiente para os usuários ao avaliarem a qualidade das imagens.
- Implementar o DIQA: embarcar o *Preditor de Avaliação de Qualidade de Imagem Profunda* (DIQA), que utiliza redes neurais convolucionais CNN, na ferramenta de-

senvolvida. Esta implementação foi realizada com o auxílio do Docker Desktop, uma ferramenta amplamente utilizada na criação de aplicações e considerada adequada para hospedar ferramentas de aprendizado profundo, conforme descrito por Xu e Chu (2017).

- Desenvolver uma segunda versão da ferramenta com o BRISQUE: documentar e implementar os procedimentos necessários para hospedar simultaneamente um segundo algoritmo de avaliação de qualidade de imagens (como o BRISQUE), não baseado em redes convolucionais.

- Analisar o desempenho da ferramenta ao ser executada diretamente sobre um ambiente virtual, em comparação à utilização do Docker Desktop.

1.3 ORGANIZAÇÃO DO TRABALHO

Esta dissertação é organizada em quatro capítulos. Além deste capítulo introdutório, temos:

- Capítulo 2: Introduz a tecnologia adotada para hospedar o algoritmo de avaliação automática de imagens, o Docker. Este capítulo detalha as etapas necessárias para iniciar a ferramenta, descreve todos os arquivos que a compõem e apresenta em detalhes o método de avaliação de qualidade de imagens sem referência incorporado na ferramenta.

- Capítulo 3: Descreve o ambiente da máquina utilizado para hospedar a ferramenta, explica como a ferramenta é manuseada através de sua interface *web* e detalha as edições necessárias para hospedar um segundo método para avaliação de qualidade de imagens. Também apresenta os resultados obtidos com a utilização da IQA Tool nestes dois cenários.

- Capítulo 4: Fornece uma análise comparativa de desempenho da ferramenta ao implementá-la diretamente em um ambiente virtual, sem utilizar o Docker. E pontua as considerações finais sobre os principais tópicos abordados nesta dissertação, incluindo as contribuições alcançadas e as sugestões para trabalhos futuros.

Essa organização visa proporcionar uma visão clara e estruturada do desenvolvimento e implementação da ferramenta de avaliação de qualidade de imagens, a IQA Tool, bem como dos resultados obtidos e das considerações finais sobre o trabalho realizado.

2 METODOLOGIA

A metodologia utilizada para criação da IQA Tool é composta por quatro elementos principais, distribuídos nas seções deste capítulo. A Seção 2.1 apresenta o Docker Desktop, a ferramenta selecionada para hospedar a IQA Tool. Esta escolha foi motivada pelo contexto específico do grupo de pesquisa ao qual a autora está vinculada, que também auxiliou na compactação dos códigos para agilizar o processo de deploy da aplicação, conforme detalhado neste trabalho.

A Seção 2.2 trata das etapas necessárias para dispor dessa aplicação, incluindo o download do Docker Desktop. A Subseção 2.2.1 lista a execução das linhas de comando necessárias para iniciar a ferramenta e, na Subseção 2.2.2, está descrito como configurar a persistência de dados na aplicação.

A Seção 2.3 exibe cada um dos arquivos compartilhados para construir a aplicação localmente.

A quarta e última seção explica o método de avaliação da qualidade de imagens embarcado na IQA Tool, desenvolvido por Kim, Nguyen e Lee (2019), o DIQA.

2.1 DOCKER

O Docker é uma tecnologia de virtualização de contêineres, conforme descrito em Anderson (2015), criada visando oferecer um recurso computacional mais leve e ágil em comparação com as máquinas virtuais VMs tradicionais. Sua utilização tem por intuito solucionar o problema de portabilidade de aplicações entre os ambientes de desenvolvimento e produção.

Quanto às arquiteturas, o autor Anderson (2015) destaca algumas das principais diferenças entre elas e fornece uma descrição geral das arquiteturas de Máquinas Virtuais VMs e do Docker para fins de comparação. As VMs são, em essência, cópias completas de sistemas operacionais que operam sobre um *hypervisor*, o qual está instalado diretamente no *hardware*. Em cima dessa pilha, as aplicações são executadas.

Por outro lado, a arquitetura do Docker adota uma abordagem diferente, ao colocar um *hypervisor* diretamente sobre o sistema operacional. Isso permite a criação de contêineres que contêm as aplicações, com um tempo de inicialização de milissegundos.

Para introduzir os termos frequentemente associados ao Docker, apresentamos a definição fornecida por Stoneman (2019), onde são utilizados os principais termos técnicos: imagem e contêiner. A imagem Docker é, essencialmente, um conjunto de componentes necessários para a execução de um programa. Ela é um artefato móvel que pode ser compartilhado e extraído de repositórios Docker. O contêiner, por sua vez, é criado quando essa imagem é iniciada e está em execução na máquina local do usuário.

Ainda segundo Anderson (2015), a arquitetura do Docker possibilita a execução de múltiplos contêineres em uma única máquina física ou virtual, oferecendo escalabilidade para as aplicações e garantindo que os *softwares* construídos no ambiente de desenvolvimento funcionarão também nos ambientes de operações e produção. Além disso, os autores Špaček, Sohlich e Dulík (2015) destacam que o Docker permite aos usuários limitar os recursos de memória RAM e CPU a serem utilizados pelos contêineres.

Em relação à natureza dos *softwares* hospedados pelo Docker, a ferramenta não demonstra limitações. O estudo realizado por Xu e Chu (2017) apontou viabilidade e benefícios ao hospedar ferramentas de aprendizagem profunda em seus contêineres devido à flexibilidade, leveza e habilidades de isolamento de recursos que o Docker proporciona. Entre as plataformas públicas que usam contêineres estão o IBM/Softlayer, Joyent e a Google, citadas por Bernstein (2014).

A literatura acerca do Docker descreve que seu desenvolvimento foi originalmente voltado ao sistema Linux, (STONEMAN, 2019). No entanto, atualmente o Docker consegue hospedar contêineres desenvolvidos em linguagens tanto para Linux quanto para Microsoft Windows, que funcionam de forma similar com base em seus respectivos *kernels*, segundo consta na documentação oficial do Docker (DOCKER, 2024b).

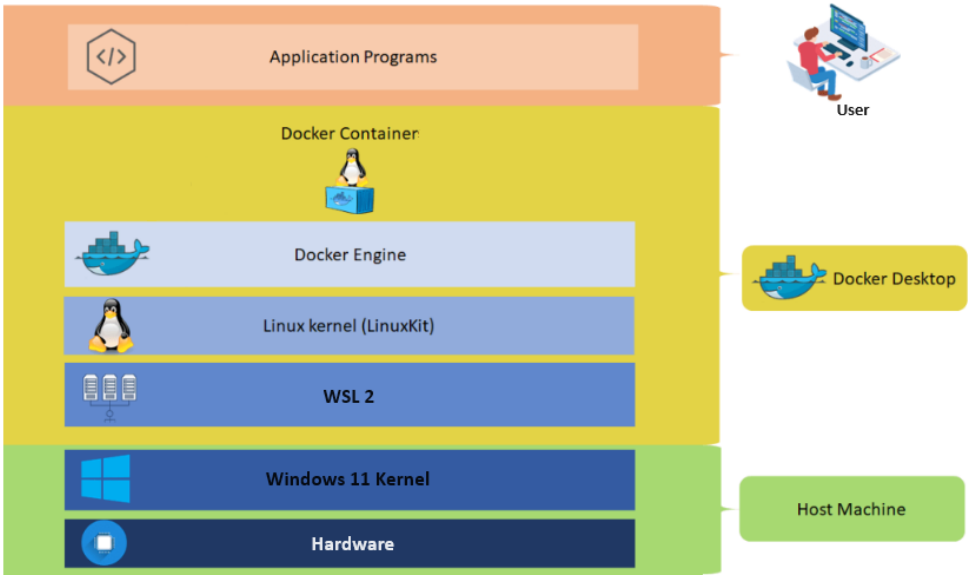
Quanto aos sistemas operacionais locais, máquinas com sistema operacional Linux restringem-se a rodar apenas contêineres Docker de mesmo *kernel*, enquanto máquinas com Microsoft Windows dispõem do *Windows Subsystem for Linux 2* (WSL 2) que fornece a integração entre contêineres Linux e o ambiente local Windows. Por meio dessa ferramenta, as máquinas Windows atualmente hospedam contêineres Docker de ambas as naturezas, tanto Windows quanto Linux, por meio do Docker Desktop.

A Figura 4 ilustra a configuração de máquina utilizada neste trabalho. Aqui, o Docker Desktop está operando sobre um computador Windows, o contêiner criado que hospeda a aplicação IQA Tool está construído sobre *kernels* Linux. Associado à arquitetura do Docker Desktop está o WSL 2, que garante a integração do Docker Engine com a máquina local.

Vale ressaltar que diversas arquiteturas, combinando máquinas virtuais, *hypervisors* e diferentes ferramentas de virtualização podem ser adotadas e estão retratadas na literatura com seu desempenho avaliado, como em Combe, Martin e Pietro (2016), Rad Harrison John Bhatti (2017), al. (2020), Xu e Chu (2017), Bernstein (2014), Anderson (2015), Špaček, Sohlich e Dulík (2015), Kwon e Lee (2020).

Existem poucas restrições quanto à utilização do Docker. Uma delas é que, atualmente, o Docker só oferece suporte a sistemas operacionais e processadores de 64 bits, conforme citado em Rad Harrison John Bhatti (2017). No caso de computadores Windows, o Docker Desktop só pode ser instalado em computadores com Windows 10 ou versões posteriores. Além disso, é recomendado verificar os requisitos vigentes no momento da instalação do Docker através de sua plataforma oficial (DOCKER, 2024c).

Figura 4 – Estrutura vigente neste trabalho: Docker Desktop instalado sobre máquina Windows usando o WSL 2 para dispor da IQA Tool.



Fonte: Elaborado pela Autora.

2.2 INSTALAÇÃO DA FERRAMENTA

Para construir a aplicação IQA Tool, é necessário, primeiramente, instalar o Docker Desktop a partir da sua plataforma oficial (<https://docs.docker.com/get-docker/>) e verificar o seu funcionamento. Essa verificação consiste em executar a seguinte linha de comando no Windows PowerShell:

> docker run hello-world

e verificar a exibição desta mensagem que sinaliza que o Docker está em funcionamento:

> “Hello from Docker!”

O Docker dispõe de um repositório virtual, o Docker Hub, que possibilita compartilhar imagens de contêineres entre seus usuários. Neste trabalho, no entanto, esta plataforma não é adotada. Os arquivos necessários para construir a imagem e o contêiner do Docker, localmente na máquina do usuário, estão sendo compartilhados via Github.

2.2.1 Iniciando a IQA Tool

Instalado o Docker Desktop e assegurado seu funcionamento, o usuário deve prosseguir com a construção da imagem e criação do contêiner em sua máquina local. Os arquivos necessários para estruturar toda a ferramenta estão disponíveis no Github da autora desta Dissertação: <https://github.com/MZMLPE/MSc-Tool>. Este conteúdo disponibilizado via Github é constituído por três arquivos principais:

1. Docker-compose.yml
2. IQA Tool (para DIQA)

3. readme.txt

Os arquivos 1 e 2 listados acima devem permanecer no mesmo diretório durante a etapa de instalação. Para construir a imagem do contêiner, é necessário ir ao diretório onde os arquivos 1 e 2 se encontram e executar a seguinte linha de comando no Windows PowerShell:

> docker compose build

Com isso, a imagem do contêiner é criada e todos os requisitos necessários para dispor da ferramenta são instalados na imagem Docker. Para iniciar o contêiner e dispor da IQA Tool, a seguinte linha de comando deve ser executada:

> docker compose up -d

Com este comando, o contêiner que hospeda a IQA Tool é iniciado e opera no modo destacado (*dettached*, -d), assegurando a inicialização do contêiner ao reiniciar o computador. Basta abrir o aplicativo do Docker Desktop para que o contêiner seja inicializado. Com a implementação apenas dessas duas linhas de comandos executadas no Windows PowerShell, o usuário dispõe da ferramenta em sua máquina local e a acessa via "localhost:80" em seu navegador.

Caso o usuário faça edições na IQA Tool e deseje hospedar mais de uma imagem ou contêiner, os seguintes comandos o auxiliarão. Para verificar os contêineres criados e seu *status* de atividade, utilize:

> docker ps

Para remover os contêineres pausados ou em desuso, o usuário deve utilizar o comando:

> docker system prune

Os contêineres que estão em execução não são afetados por esse comando. Para verificar as imagens criadas, execute:

>docker images

Com esse comando é possível verificar o ID das imagens presentes no Docker. Para remover as imagens em desuso, o usuário deve utilizar o comando:

> docker rmi image_id

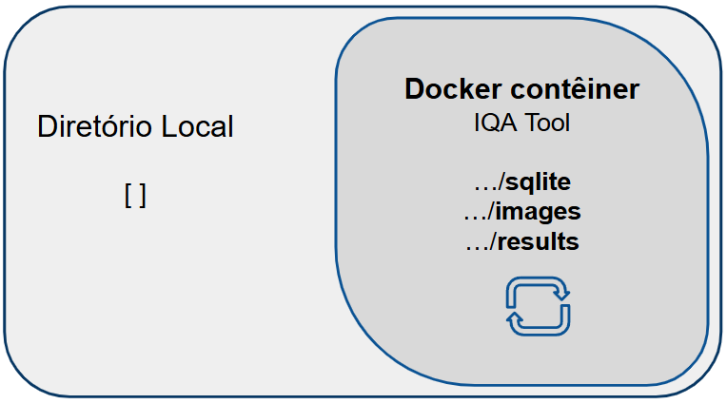
Dessa forma, não haverá acúmulo de resíduos e a sobrecarga de memória é evitada.

2.2.2 Configuração da Persistência de Dados

A persistência de dados no Docker refere-se à capacidade de manter os dados dos contêineres armazenados, mesmo após o contêiner ser reiniciado ou removido. Conforme exposto na documentação oficial do Docker, (DOCKER, 2024e), a configuração da persistência de dados pode ser realizada empregando o chamado Docker Volumes. Ao configurar a persistência de dados utilizando o Docker Volumes, o *backup* e a restauração dos dados do contêiner ficam assegurados. A sua utilização permite também que múltiplos contêineres compartilhem dados.

À medida que o usuário utiliza a IQA Tool, apenas o diretório interno do contêiner é atualizado, nele estão hospedados o banco de dados "db.sqlite", as pastas com as imagens inseridas e planilhas de resultados obtidos, dentre outros componentes da ferramenta. Conforme elucidado na documentação oficial do Docker, (DOCKER, 2024d), por padrão, a criação, modificação e remoção desses arquivos internos a um contêiner não afeta outros contêineres ou o sistema local. E qualquer dado escrito em um contêiner é perdido quando o contêiner é parado ou removido, já que os contêineres operam de forma isolada, conforme ilustra a Figura 5.

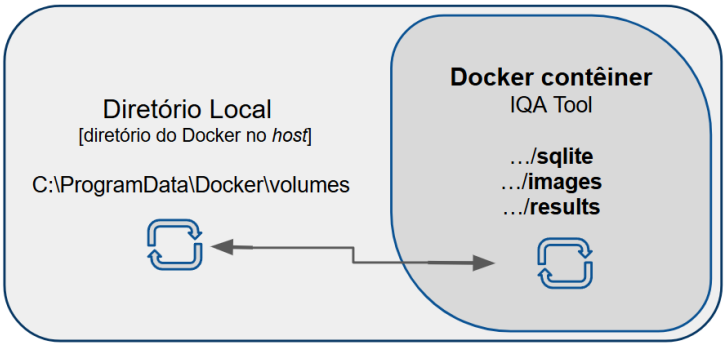
Figura 5 – Antes da criação do volume local: pastas e arquivos sendo atualizados dentro do contêiner durante a utilização da ferramenta.



Fonte: Elaborado pela autora.

Neste cenário, caso o contêiner seja deletado ou precise ser reconstruído por algum motivo, as atualizações realizadas nos arquivos citados serão perdidas. A utilização dos chamados *volumes do Docker* garante a persistência desses dados ao reconstruir este contêiner. Com a utilização dos volumes, o Docker vincula o diretório interno do contêiner a um diretório local e mantém esses arquivos atualizados, conforme ilustra a Figura 6.

Figura 6 – Após a criação do volume local: pastas e arquivos sendo atualizados durante a utilização da ferramenta. Vinculadas ao diretório local.



Fonte: Elaborado pela autora.

Para empregar o Docker Volumes de forma prática, basta configurá-lo no arquivo "docker-compose.yml". Seguindo a explicação apresentada em Docker (2024a) e Docker (2024f), as linhas 9 em diante foram implementadas na IQA Tool com este propósito:

- **docker-compose.yml**

```
1.  version: '3.7'
2.  services:
3.      iqa_diqa:
4.          build: ./IQA Tool DIQA
5.          container_name: iqa_diqa
6.          restart: always
7.          ports:
8.              - '80:8080'
9.          volumes:
10.             - diqa_tool_dados : /app/app/sqlite
11.             - diqa_tool_images : /app/app/images
12.             - diqa_tool_results : /app/app/results
13. volumes:
14.     diqa_tool_dados:
15.     diqa_tool_images:
16.     diqa_tool_results:
```

As linhas 9, 10, 11 e 12 tratam da criação do Docker Volume e da atribuição dos nomes às pastas a serem criadas na máquina local para persistência dos dados e as vinculam ao diretório do contêiner onde constam as pastas a serem preservadas. As linhas 13, 14, 15 e 16 fazem menção à lista dos volumes já criados e às pastas que serão utilizadas como referência no momento de criação do contêiner. Desta forma, o Docker verifica a existência das pastas armazenadas no diretório local a serem utilizadas como referência na criação do contêiner e as mantêm atualizadas em sincronia com os arquivos do contêiner em operação.

Realizando a inserção destas linhas no "docker-compose.yml", basta salvar as alterações e iniciar a ferramenta normalmente, como descrito na Seção 2.2.1. Apesar de não ser o caso utilizado neste trabalho, essa configuração é empregada também para compartilhar dados entre contêineres distintos em operação.

Para verificar os volumes criados pelo Docker, utiliza-se o comando:

> docker volume ls

Caso o usuário deseje excluir o volume local criado pelo Docker e interromper a sincronia dos arquivos, basta executar:

> docker volume rm volume_name

2.3 ESTRUTURA DE PASTAS E ARQUIVOS DA IQA TOOL

A IQA Tool tem suas configurações de *frontend* e *backend* definidas e usa o Flask, o qual é um conjunto de ferramentas do Python (*Python framework*) utilizado para desenvolvimento *web* (DWYER, 2016, GRINBERG, 2018). Com o Flask, obtivemos uma interface clara e objetiva, de fácil manuseio. A ferramenta também embarca o banco de dados SQLite (ALLEN, 2010) para armazenamento dos dados de acesso dos usuários e registro de execuções realizadas.

Em seu código principal, "source.py", constam as etapas de leitura, pré-processamento das imagens, avaliação e geração da planilha de resultados. Para avaliação das imagens, a ferramenta usa a rede neural convolucional que consta no arquivo "model.h5", obtida pela implementação do método de avaliação automática de imagens DIQA Kim, Nguyen e Lee (2019), descrito na próxima seção deste capítulo.

O arquivo "requirements.txt" contém todas as dependências necessárias para a execução correta da aplicação. Nele estão especificadas as versões das bibliotecas necessárias para hospedar a aplicação em um ambiente novo, seja um ambiente virtual, uma máquina virtual ou uma nova imagem Docker.

Para o desenvolvimento da primeira versão da IQA Tool aqui exposta, um arquivo "requirements.txt" previamente criado no grupo de trabalho CIn/Motorola foi utilizado como referência. Para hospedar a segunda versão da IQA Tool, o documento "requirements.txt" foi editado manualmente para contemplar as novas necessidades da IQA Tool em sua segunda versão. Foi realizada a adição do pacote "brisque" à lista de requisitos.

Dentre as bibliotecas que este arquivo engloba, destacam-se as bibliotecas para criação do *framework web* e hospedagem da ferramenta, o "Flask"; o "Flask-Login", para o gerenciamento de sessões; "Flask-SQLAlchemy", para integração com o banco de dados; "uWSGI", responsável pela configurações da hospedagem da aplicação e "Werkzeug", que é uma de suas bibliotecas auxiliares.

Para atender aos quesitos relacionados ao aprendizado de máquina e aprendizado profundo foram utilizadas as bibliotecas "Keras", "TensorFlow", "tensorflow-estimator" e "tensorboard", também a "opt-einsum", "absl-py", "gast" e "libclang" para o suporte interno e otimizações destas bibliotecas.

Nos componentes de processamento de imagens, estão presentes as bibliotecas "Scikit-Image", "Pillow" e "opencv-python" para fornecer as utilidades voltadas à visão computacional e "imageio", "PyWavelets" e "tifffile" para manipulação de arquivos de alta resolução. As bibliotecas "numpy", "scipy" e "networkx" estão auxiliando na manipulação de dados e realização de cálculos da ferramenta. E os pacotes "SQLAlchemy" e "greenlet", no manuseio do banco de dados utilizado pela ferramenta.

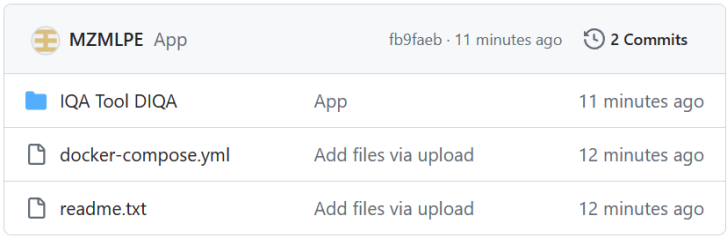
Para autenticação e segurança, estão sendo utilizadas as bilbiotecas "google-auth", "oauthlib", "rsa" e "itsdangerous", além da "pyasn1" e "pyasn1-modules", para comunicações seguras. As bibliotecas de requisições HTTP e manipulação de arquivos incluem

“requests”, “urllib3”, “certifi”, “idna”, “charset-normalizer” e “zipp”. Por fim, as ferramentas de desenvolvimento e utilitários solicitaram o uso das bibliotecas “click”, “Markdown”, “MarkupSafe”, “packaging”, “termcolor”, “XlsxWriter”, “wrapt”, “importlib-metadata” e “typing_extensions”.

Para a criação de um novo “requirements.txt”, recomenda-se que o desenvolvedor desenvolva sua aplicação em um ambiente virtual isolado e, após concluir toda a implementação, gere este arquivo através do comando “pip freeze > requirements.txt”, que lista todas as bibliotecas necessárias juntamente com suas versões específicas, conforme indicam Hunt (2023) e Oort et al. (2021). Esse processo é essencial para garantir um *deploy* eficiente, e assegura que o ambiente de produção inclua todas as dependências indispensáveis para a execução correta da ferramenta em outros ambientes.

Como citado anteriormente, todos os arquivos necessários para construir a IQA Tool estão disponibilizados através do Github. Esta seção elucida esses componentes e a hierarquia sob a qual estão organizados. A Figura 7 exibe os três arquivos principais disponibilizados.

Figura 7 – Arquivos principais disponibilizados via Github da autora.



Fonte: Elaborado pela autora.

A partir desta estrutura estão detalhados os seus principais arquivos. A começar por:

readme.txt: onde estão as instruções para verificar a correta operação do Docker, os comandos para construir a imagem e iniciar o contêiner da IQA Tool, e sua porta de acesso.

- docker-compose.yml: arquivo sobre o qual os comandos principais se baseiam para construção da imagem e inicialização do contêiner.

```
1. version: '3.7'
2. services:
3.   iqa_diqa:
4.     build: ./IQA Tool DIQA
5.     container_name: iqa_diqa
6.     restart: always
7.     ports:
8.       - '80:8080'
```

Neste arquivo, a linha 1 é responsável por identificar a versão do “docker-compose.yml” em uso. O comando da linha 2 inicia a declaração dos serviços que serão criados com o Docker. A linha 3 nomeia a rede criada para o contêiner e a linha 4 aponta o endereço da pasta referência para construção da imagem do contêiner. A linha 5 designa o nome a ser utilizado pelo contêiner. O comando descrito na linha 6 autoriza ao Docker reiniciar o contêiner automaticamente caso ele pare e as linhas 7 e 8 fazem a alocação de portas entre *host* e contêiner.

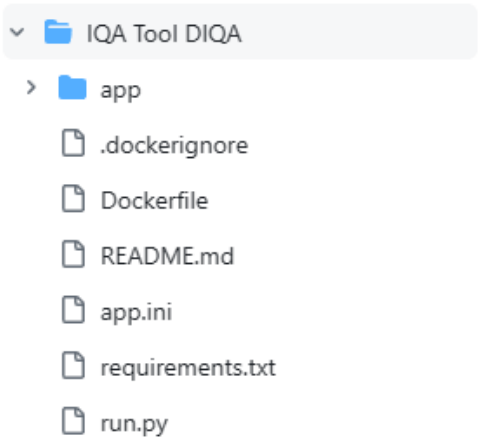
Vale salientar que essa versão do “docker-compose.yml” não incorpora a construção de volumes para persistência de dados. Conforme exposto anteriormente em 2.2.2, caso o usuário deseje implementar esse setor, basta inserir algumas linhas a este arquivo e prosseguir normalmente com as etapas de inicialização da ferramenta descritas em Iniciando a IQA Tool, 2.2.1.

A seguir está a composição da pasta principal compartilhada para construção da IQA Tool, pasta “IQA Tool DIQA”, ilustrada na Figura 8. Conforme citado anteriormente, seu conteúdo engloba arquivos .html para configuração da interface *web*, .sqlite (banco de dados), pastas de imagens, planilhas e outros. Alguns dos arquivos que compõem a pasta “IQA Tool DIQA” estão abertos para exibição nesta seção, os demais constam no Anexo A.

Vale ressaltar que, no capítulo de experimentos, estão detalhados quais destes arquivos requerem modificações para hospedar outro algoritmo para avaliação de imagens (no caso, o BRISQUE).

- IQA Tool DIQA:

Figura 8 – Arquivos principais disponibilizados via Github.



Fonte: Elaborado pela autora.

run.py: comando para inicialização do aplicativo.

requirements.txt: Arquivo que contém todas as dependências necessárias para executar a aplicação corretamente. Lista de bibliotecas a serem instaladas na imagem Docker para configurar o ambiente da ferramenta.

app.ini: realiza a alocação de portas, e quantidade de processos e *threads* necessárias para iniciar o serviço.

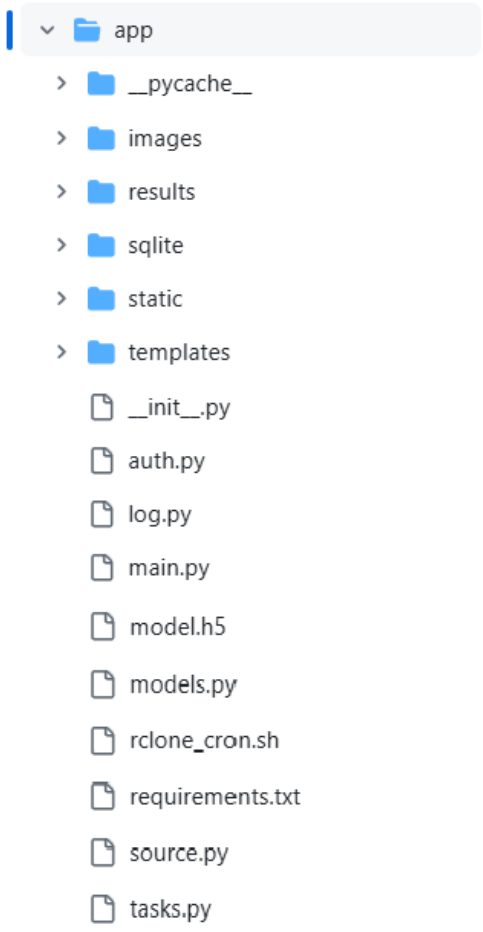
README.md: contém instruções de manuseio do Docker.

Dockerfile: arquivo padrão do Docker utilizado para construção da imagem que chamada para a lista dos requisitos.

.dockerignore: especifica os arquivos que não serão incluídos na imagem construída pelo Docker.

app: esta pasta hospeda algumas das principais definições da ferramenta, entre elas, a rede neural treinada (arquivo .h5), e o código "source.py".

Figura 9 – Pasta app, componentes da IQA Tool.



Fonte: Elaborado pela autora.

Concluída a passagem sobre os componentes da pasta "IQA Tool DIQA", seguimos com os principais componentes da sua subpasta "app", ilustrada na Figura 9.

- app:

  tasks.py: onde é definida a função de inicialização da execução do algoritmo.

  source.py: embarca etapas de pré-processamento, avaliação da imagem e confecção das planilhas com resultados da avaliação.

  requirements.txt: listagem de algumas bibliotecas necessárias para execução da ferramenta.

  rclone_cron.sh: arquivo para sincronizar diretório remoto ao diretório local. Atualmente, não influencia no funcionamento da ferramenta.

  models.py: faz a leitura dos usuários e das execuções contidas no banco de dados .sqlite e nomeia seus componentes.

  model.h5: CNN treinada sobre o *dataset* Live para previsão de notas de imagens sem referência.

  main.py: define as funções disponíveis ao usuário na interface *web* e as relaciona com os comandos do processo interno (*backend*).

  log.py: código que define a função "LOG" utilizada para exibir no terminal do Docker mensagens do *status* do funcionamento da ferramenta.

  auth.py: definições da interface *web* relacionadas à edição e adição de usuários.

  ___init___: inicia a estrutura de criação e inicialização da aplicação. Realiza a importação do Flask. Declara da *Secret Key*.

  /templates: pasta com os arquivos .HTML, utilizados para definição da aparência da ferramenta e interação dos usuários.

  /static: figuras auxiliares para exibição na interface *web*.

  /sqlite: banco de dados contendo planilha de usuários e execuções da ferramenta.

  /results: pasta com as planilhas de resultados obtidos pela ferramenta.

  /images: pasta com as imagens inseridas da ferramenta.

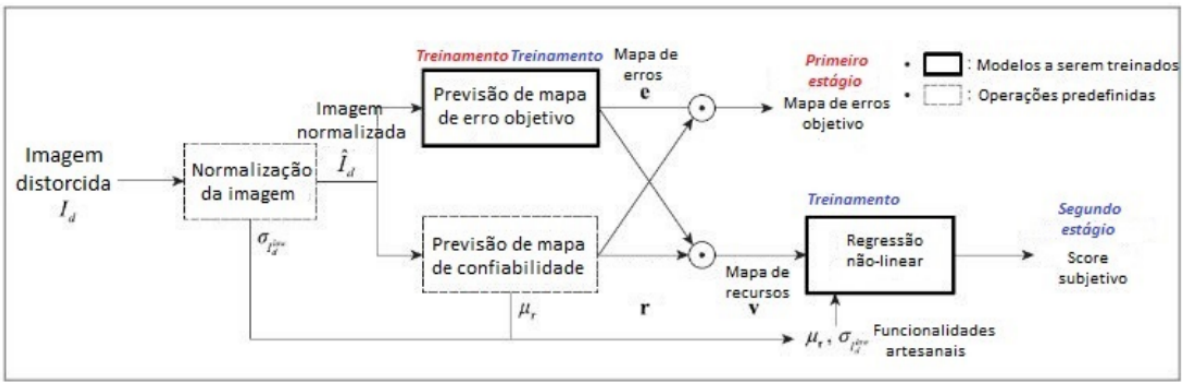
  /___pycache___: cache com arquivos compilados de códigos python no formato .cpython-38.

2.4 PREDITOR DE AVALIAÇÃO DE QUALIDADE DE IMAGEM PROFUNDA

Optamos por incorporar à ferramenta o algoritmo *Preditor de Avaliação de Qualidade de Imagem Profunda* (DIQA), introduzido por Kim, Nguyen e Lee (2019), por se tratar de um modelo propício a refinamentos adicionais, que permite otimizações conforme o seu propósito de utilização, como já realizado pelo grupo de trabalho do CIn/Motorola. Sua estrutura é exibida na Figura 10, onde a imagem de entrada é normalizada e passa por

dois caminhos: 1) um ramo de CNN e 2) um ramo de previsão de mapa de confiabilidade. No primeiro estágio do treinamento, o ramo de CNN é treinado para prever um mapa de erro objetivo e. No segundo estágio, o modelo é treinado adicionalmente para prever uma pontuação subjetiva humana S. Em cada estágio, o mapa de confiabilidade r é suplementado para compensar a imprecisão em regiões homogêneas.

Figura 10 – Diagrama de fluxo geral do DIQA. O processo de treinamento consiste em duas etapas: regressão nos mapas de erro objetivos e regressão nas pontuações subjetivas. Os blocos com "treino" em vermelho (ou "treino" em azul) indicam que a sub-rede será treinada na primeira (ou segunda) etapa.

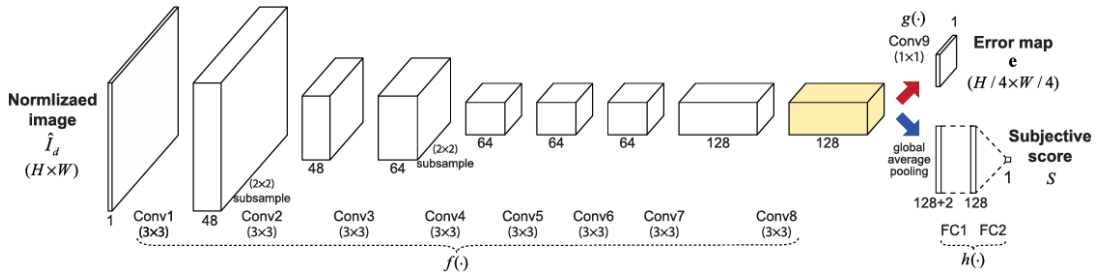


Fonte: (KIM; NGUYEN; LEE, 2019) adaptado por SILVA (2022).

2.4.1 Arquitetura do Modelo

O design da arquitetura de *Redes Neurais Convolucionais* (CNN) proposta é motivado por Chen et al. (2020) e exibido na Figura 11.

Figura 11 – Arquitetura da rede de previsão do mapa de erro de objetivo *pixel*-a-*pixel*. "Conv" indica as camadas de convolução e "FC" indica as camadas totalmente conectadas. O texto abaixo de "Conv" indica o tamanho do filtro. As setas vermelhas e azuis indicam os fluxos da primeira e segunda etapa respectivamente.



Fonte: (KIM; NGUYEN; LEE, 2019).

Na primeira etapa, a qual é a previsão do mapa de erros, o modelo consiste apenas em camadas de convolução, onde as bordas são preenchidas com zeros antes de cada

convolução. Desta forma, as saídas não perdem informações referentes à posição relativa do *pixel*.

Cada camada, exceto a última, possui um filtro 3×3 e uma Unidade Linear Retificada RELU (NAIR; HINTON, 2010). Para a operação de *downsampling*, é usada uma convolução com *stride* de 2. A saída do Conv8 é chamada de mapa de características e está identificada em amarelo na Figura 11. Este mapa de características é reutilizado para o segundo estágio de treinamento.

Na segunda etapa de treinamento, o mapa de características extraído é alimentado na camada de *pooling* médio global *Global Average Pooling* (GAP), seguido pelas camadas FC1 e FC2 apontadas na Figura 11. Os procedimentos de Conv1 a Conv8 são denotados por $f(\cdot)$, a operação do Conv9 por $g(\cdot)$, e o procedimento incluindo FC1 e FC2 por $h(\cdot)$, conforme ilustra a Figura 11.

2.4.2 Normalização da Imagem

Como pré-processamento, as imagens de entrada são primeiro convertidas em escala de cinza e subtraídas da própria imagem, após filtrada por um filtro passa-baixa. Seja I_r uma imagem de referência e I_d a imagem distorcida correspondente. As versões normalizadas são então denotadas por $\hat{I}_r$ e $\hat{I}_d$, respectivamente. A imagem de baixa frequência é obtida, reduzindo a imagem de entrada para 1/4 e aumentando-a novamente para o tamanho original, denotado por I_r^{low} e I_d^{low}. Um filtro passa-baixa Gaussiano e subamostragem são usados para redimensionar as imagens.

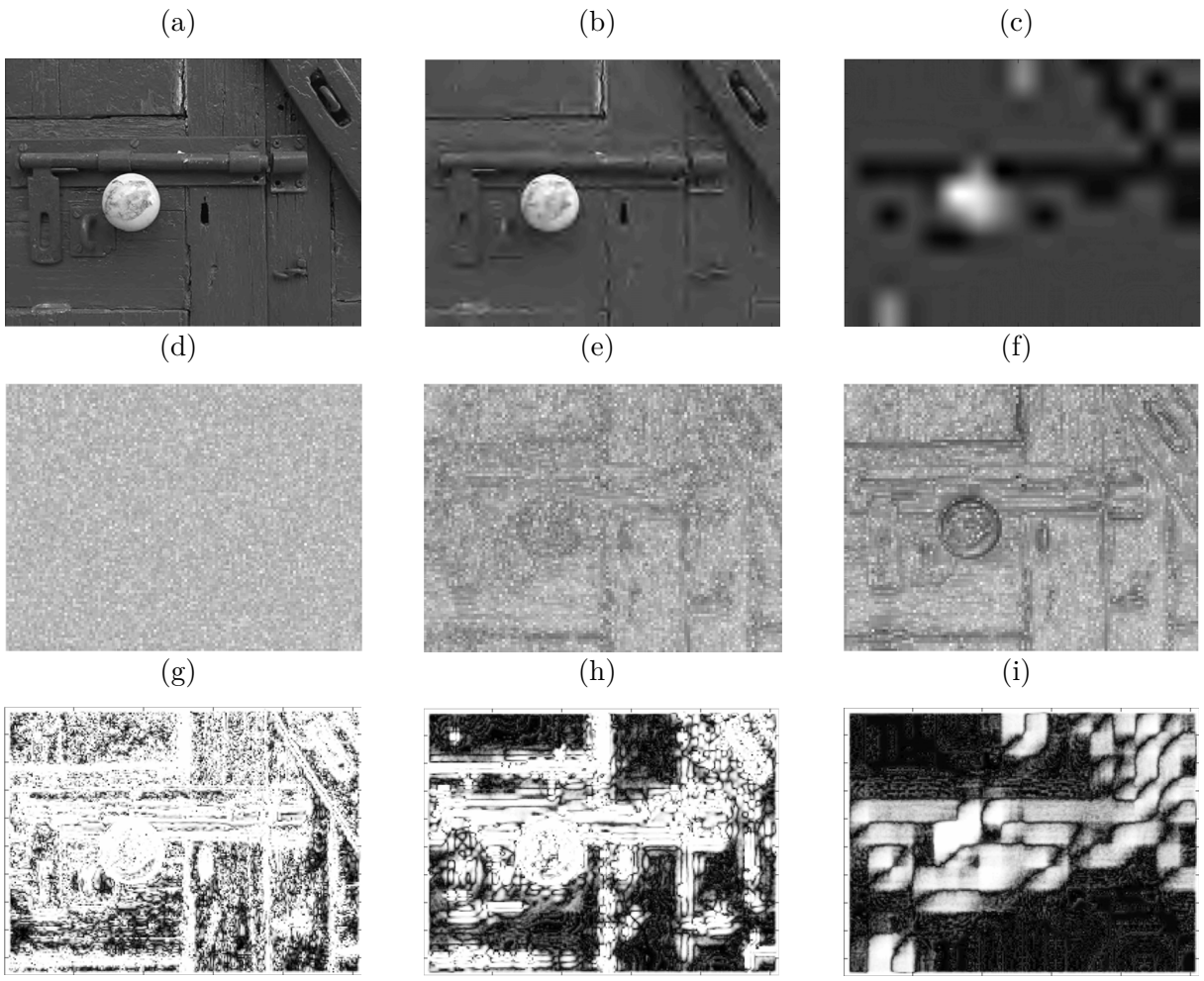
2.4.3 Previsão do Mapa de Confiabilidade

Muitas distorções, como quantização por JP2K ou *Gaussian Blurr* (GB), tornam as imagens borradas. No entanto, ao contrário de algoritmos FR-IQA, é difícil determinar se uma região borrada está distorcida sem conhecer a imagem original. Além disso, à medida que uma distorção severa é aplicada a uma imagem, seu mapa de erro recebe mais componentes de alta frequência. Enquanto isso, a imagem distorcida perde mais detalhes de alta frequência, como mostrado na Figura 12. Portanto, é provável que o modelo falhe em prever o mapa de erro objetivo em regiões homogêneas.

Para evitar esse problema, a confiabilidade do mapa de erro previsto é estimada, medindo a presença de textura da imagem distorcida. Os autores Kim, Nguyen e Lee (2019) assumem que regiões borradas têm menor confiabilidade do que regiões texturizadas, já que essas apresentam mais detalhes. Imagens pré-processadas que passam por um filtro passa-faixa são usadas para medir o mapa de confiabilidade r, como na equação 2.1:

$$r = \frac{2}{1 + \exp(-\alpha|\hat{I}_d|)} - 1 \tag{2.1}$$

Figura 12 – Exemplos de mapas de confiabilidade estimados. (a)–(c) Imagens distorcidas por JPEG2000 no conjunto de dados TID2013 nos níveis de distorção 1, 3 e 5. (d)–(f) Mapas de diferença derivados. (g)–(i) Mapas de confiabilidade de (a)–(c).



Fonte: Extraído de Kim, Nguyen e Lee (2019).

onde α controla a saturação do mapa de confiabilidade e $\hat{I}_d$ é a imagem distorcida normalizada. Para normalizar o mapa de confiabilidade, a metade positiva da função sigmóide é usada na equação 2.1, de modo que *pixels* com valores pequenos recebam valores de confiabilidade altos.

As imagens mostradas nas Figuras 12 (a)–(c) são distorcidas pelo padrão JPEG2000 em diferentes níveis, e os mapas de confiabilidade correspondentes com α=1 são mostrados na Figura 12 (d)–(f). Pode-se verificar que é difícil derivar um mapa de erro preciso (Figura 12(f)) a partir de imagens severamente distorcidas (Figura 12(c)). Os mapas de confiabilidade estimados são mostrados na Figura 12(g)–(i). Como mostrado na Figura 12 (i), o mapa de confiabilidade tem valores zero, onde não há informação espacial significativa na Figura 12 (c).

Para evitar que o mapa de confiabilidade afete diretamente a pontuação prevista, ele é dividido pela sua média e passa a ser chamado $\hat{r}$:

$$\hat{r} = \frac{1}{\frac{1}{(H_r \cdot W_r)} \sum_{i,j} r(i,j)} \cdot r \tag{2.2}$$

onde H_r e W_r são a altura e largura da imagem.

2.4.4 Aprendizado do Mapa de Erro Objetivo

Na primeira etapa de treinamento, os mapas de erro objetivos são usados como alvos de regressão com *proxies* para obter o efeito de aumento de dados. A função de perda L_1 é definida pelo erro quadrático médio entre os mapas de erro previstos e os mapas de erro verdadeiros.

2.4.5 Aprendizado de Opinião Subjetiva

Treinado o modelo para prever os mapas de erro objetivos, a metodologia segue para a próxima etapa de treinamento, onde o DIQA é treinado para prever as pontuações subjetivas. Para isso, a sub-rede treinada $f(\cdot)$ é conectada a uma camada de *pooling* médio global seguido pelas camadas totalmente conectadas, conforme mostrado na Figura 11. O mapa de características é medido sobre o domínio espacial, resultando em um vetor de características de 128 dimensões.

Nesta etapa, para compensar a informação perdida, são adicionadas duas características ao modelo: a média do mapa de confiabilidade não normalizado μ_r e o desvio padrão da baixa frequência da imagem distorcida $\sigma_{I_d^{low}}$.

2.4.6 Treinamento baseado em *patches*

Na arquitetura do DIQA, os tamanhos das imagens de entrada devem ser fixos para treinar o modelo em uma GPU. Portanto, para treinar o DIQA, usando imagens de vários tamanhos, como no banco de dados LIVE IQA (SHEIKH; SABIR; BOVIK, 2006), cada imagem de entrada foi dividida em vários blocos de mesmo tamanho. O passo da janela deslizante é determinado pelo tamanho do bloco e pelo número de *pixels* ignorados ao redor das bordas para evitar regiões sobrepostas quando o mapa de erro perceptual é reconstruído. Quando os *pixels* ignorados ao redor das bordas são quatro, o passo deve ser 4, onde $step_{patch} = size_{patch} - 32$ é determinado por 4×2 (ambos os lados da borda) $\times 4$ (ampliação por 4). No experimento com o banco de dados LIVE IQA, o tamanho do bloco era 112×112 e cada passo era 80×80.

Além disso, durante o treinamento da segunda etapa, todos os *patches* que compõem uma imagem foram mantidos no mesmo mini-lote (KIM; LEE, 2017), de modo que v, μ_r e $\sigma_{I_{\mathrm{low}d}}$ foram derivados dos mapas de erro perceptual e de confiabilidade reconstruídos.

3 EXPERIMENTOS E RESULTADOS

Este capítulo apresenta como foram realizados os *deploys* da IQA Tool com duas configurações distintas: primeiro hospedando o algoritmo DIQA, que usa de uma rede neural convolucional; segundo, hospedando o algoritmo BRISQUE (MITTAL; MOORTHY; BOVIK, 2012) que não usa redes neurais e também difere nas etapas de pré-processamento de imagens. Na Seção 3.5 estão destacados os arquivos envolvidos nas alterações necessárias do código base disponibilizado no Github para hospedar a segunda versão da IQA Tool, fundamentada sobre o BRISQUE.

Seguindo o passo-a-passo descrito no Capítulo 2, cumprem-se as etapas necessárias para instalação da IQA Tool: a instalação do Docker Desktop, o *download* dos arquivos referentes à IQA Tool através do Github e a construção da imagem e inicialização do contêiner no Docker Desktop. As propriedades da máquina utilizada para realização dos experimentos estão descritas a seguir, na Seção 3.1.

O usuário que deseja utilizar a IQA Tool deve considerar que o tempo de construção da imagem no Docker para disponibilizar a ferramenta pode variar, dependendo dos recursos disponíveis na máquina em uso. Além disso, o tempo de execução do algoritmo embarcado na ferramenta depende da quantidade de imagens submetidas à avaliação.

Nas seções seguintes estão detalhadas as informações referentes à máquina utilizada para realização dos experimentos, o banco de dados de onde as imagens experimentais foram extraídas, as modificações necessárias para dispor da IQA Tool com as duas configurações simultaneamente. Na Seção 3.6 estão os resultados obtidos com a IQA Tool na avaliação das imagens inseridas utilizando os dois métodos distintos: o DIQA e o BRISQUE.

Ao final deste capítulo, na Subseção 3.6.2, apresenta-se a análise do tempo de processamento da ferramenta ao avaliar 10 imagens com dimensões de 2.880x2.160 pixels em distintos cenários: com ou sem a utilização do Docker, e com a hospedagem do DIQA ou BRISQUE na ferramenta. Para cada um dos cenários, os experimentos foram repetidos 30 vezes, para uma maior validação estatística.

3.1 AMBIENTE

A máquina utilizada para realizar este experimento é um notebook com sistema operacional Windows 11. Suas configurações estão descritas na Tabela 1. Vale ressaltar que máquinas com processador inferior a $\times 64$ bits, ou memória RAM inferior a 4GB, não conseguem suportar o Docker.

Tabela 1 – Configurações da máquina utilizada na execução dos experimentos

Especificações do Windows	Windows 11
Tipo de sistema	Sistema operacional de 64 bits
Processador	AMD Ryzen 7 3700U com Radeon Vega Mobile Gfx, 2.30 GHz, 4 Cores
Memória RAM utilizável/instalada	9,94/12 GB
SSD	475 GB
Docker instalado	Docker Desktop 4.25.2

Fonte: Elaborado pela autora.

3.2 DADOS DE ACESSO

A IQA Tool está sendo disponibilizada via Github com configurações de acesso genéricas. Iniciado o contêiner, o usuário então dispõe da ferramenta no seu navegador padrão, e a acessa pela porta local <localhost:80>. Dois *logins* estão configurados para acesso à ferramenta: um acesso como Administrador e um acesso como Usuário Comum (Tabela 2). Apenas o Administrador consegue editar a rede neural utilizada pelo modelo, se for o caso do modelo requerer uma rede neural para uso. O Usuário Comum dispõe de uma interface reduzida, não sendo permitido, por exemplo, inserir ou editar o algoritmo de avaliação utilizado.

Tabela 2 – Usuários cadastrados

Administrador	
Email:	admin@iqatool.com
Nome:	Admin
Senha:	admin
Usuário Comum	
Email:	user@iqatool.com
Nome:	User 01
Senha:	user01

Fonte: Elaborado pela autora.

3.3 BANCO DE DADOS HRIQ

As imagens submetidas à avaliação na IQA Tool foram extraídas do banco de dados público *High Resolution Image Quality* (HRIQ), apresentado por Huang Qiang Wan

(2024). Este banco de dados contém 1.120 imagens, disponibilizadas em três dimensões diferentes. As imagens originais têm dimensões 2.880x2.160 pixels e foram reduzidas às dimensões 1.024x768 pixels e 512x384 pixels.

O estudo conduzido pelo autor do HRIQ mobilizou um total de 175 participantes com o propósito de avaliar as imagens de maior resolução e atribuir a elas notas subjetivas. Cada uma das imagens originais foi avaliada por cerca de 25 indivíduos, e as notas atribuídas passaram por um processo de refinamento para remover avaliações enviesadas.

Com o propósito de analisar a capacidade de processamento das duas versões da IQA Tool hospedadas no Docker Desktop sobre a máquina de configurações descritas na Seção 3.1, foram selecionadas imagens de resoluções distintas oriundas do HRIQ. O trabalho aqui desenvolvido não tem por objetivo analisar a coesão entre as notas obtidas pelos avaliadores humanos do estudo citado e as notas obtidas pela IQA Tool utilizando os métodos DIQA e BRISQUE.

A Tabela 3 contém as 20 imagens de resolução de 512x384 pixels usadas para os testes.

3.4 EXPERIMENTO 1: *DEPLOY* DO DIQA

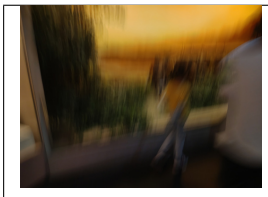
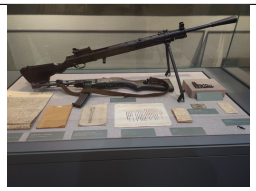
A imagem construída no Docker para hospedar os requisitos solicitados pela ferramenta e ambientar suas configurações de *backend* ocupa uma memória de 4,04GB. Conforme citado anteriormente, a IQA Tool não inclui a etapa de treinamento do DIQA, é necessário importar a rede neural convolucional já treinada para que a ferramenta a utilize. A ferramenta pode embarcar uma CNN de configurações distintas, desde que esteja no formato compatível .h5 (ou .tf).

Conforme descrito na Seção 2.4, a rede neural utilizada neste trabalho foi treinada segundo o modelo proposto por Kim, Nguyen e Lee (2019). O arquivo referente à rede neural treinada está intitulado “model.h5” e consta na pasta “app” já disponibilizada no Github, conforme exposto na Seção 2.3.

Para avaliar as imagens, os arquivos devem estar unidos em uma pasta compactada como “Pasta.zip/Pasta/ImagensXX.jpg”. O arquivo “.zip” é reconhecido pela ferramenta (Figura 13).

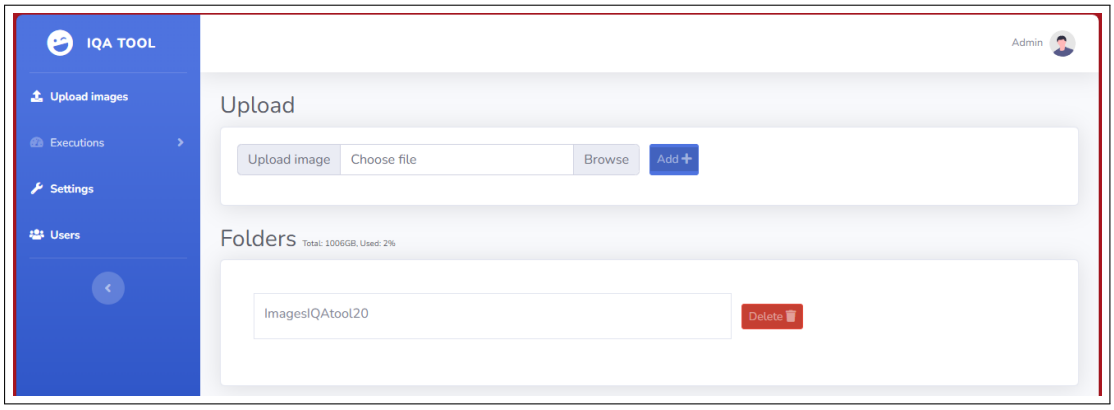
Adicionada a pasta de imagens à ferramenta, é realizada sua execução na aba “Executions/New”. A planilha com os resultados obtidos pelo método embarcado na ferramenta fica então disponível para *download* na aba “Executions/Results”, clicando em Download (ver Figura 14).

Tabela 3 – Imagens extraídas da base HRIQ inseridas na IQA Tool para predição de notas.

			
1101.jpg	1102.jpg	1102.jpg	1104.jpg
			
1105.jpg	1106.jpg	1107.jpg	1108.jpg
			
1109.jpg	1110.jpg	1111.jpg	1112.jpg
			
1113.jpg	1114.jpg	1115.jpg	1116.jpg
			
1117.jpg	1118.jpg	1119.jpg	1120.jpg

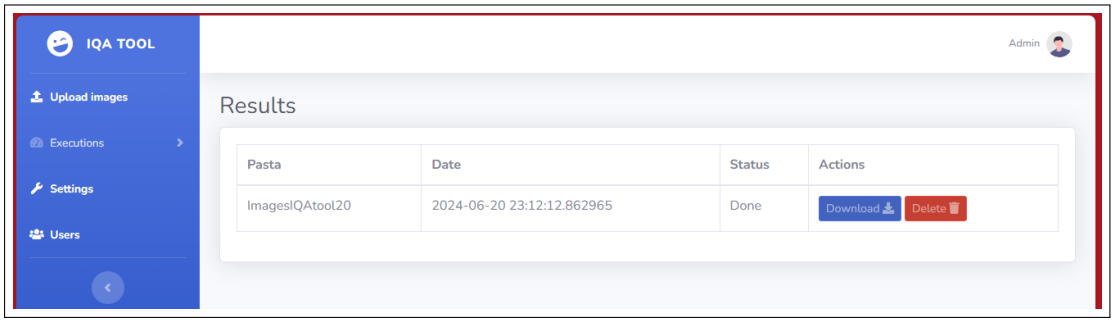
Fonte: Banco de dados HRIQ (HUANG QIANG WAN, 2024)

Figura 13 – *Upload* de arquivo .zip, contendo imagens para avaliação.



Fonte: Elaborado pela autora.

Figura 14 – Notas obtidas pela IQA Tool neste experimento.



Fonte: Elaborado pela autora.

3.5 EXPERIMENTO 2: *DEPLOY* DO BRISQUE

O BRISQUE (*Blind/Referenceless Image Spatial Quality Evaluator*) (MITTAL; MOORTHY; BOVIK, 2012) é um modelo de avaliação de imagens sem referência que opera sobre o domínio espacial das imagens e avalia a qualidade da imagem com base em estatísticas de cenas naturais (NSS - *Natural Scene Statistics*). O modelo não identifica distorções específicas, como borramento, e usa estatísticas de cena de coeficientes de luminância normalizados localmente para quantificar possíveis perdas de naturalidade na imagem devido à presença de distorções.

O BRISQUE não usa redes neurais e, em comparação ao DIQA, possui menor complexidade computacional, considerando a necessidade de treinamento do DIQA. Seu artigo original disponibiliza o algoritmo em linguagem C, sua versão em Python é utilizada neste trabalho e é oriunda do repositório oficial de pacotes para Python <https://pypi.org/project/brisque/>. As alterações necessárias para hospedá-lo estão descritas a seguir.

Conforme visto na Seção 2.3 do capítulo anterior, ilustrado na Figura 7, a construção da IQA Tool envolve principalmente um arquivo "docker-compose.yml" e a pasta "IQA Tool DIQA", onde estão os arquivos .html e as definições das bibliotecas a serem instaladas

na imagem Docker. Para hospedar o algoritmo BRISQUE, é necessário editar o arquivo "docker-compose.yml" e realizar uma cópia da pasta "IQA Tool DIQA" para então alterar alguns de seus componentes e assim obter uma segunda ferramenta que usa o BRISQUE, rodando em simultâneo com a original.

3.5.1 Edição do "docker-compose.yml"

O arquivo "docker-compose.yml", disponibilizado via Github, está configurado para iniciar a ferramenta que hospeda o DIQA. Para iniciar a nova versão da ferramenta (com o BRISQUE), este arquivo deve ser modificado nas linhas 3, 4, 5 e 8, seguindo o exposto:

- Versão do "docker-compose.yml" utilizado para iniciar a IQA Tool hospedando o DIQA:

```
1.version: '3.7'
2.services:
3.  iqa_diqa:
4.    build: ./IQA Tool DIQA
5.    container_name: iqa_diqa_final
6.    restart: always
7.    ports:
8.      - '80:8080'
```

- Versão do "docker-compose.yml" utilizado para iniciar a IQA Tool hospedando o BRISQUE:

```
1.version: '3.7'
2.services:
3.  iqa_brisque:
4.    build: ./IQA Tool BRISQUE
5.    container_name: iqa_brisque_final
6.    restart: always
7.    ports:
8.      - '82:8282'
```

Desta forma, foram modificados: a pasta de referência, o nome do contêiner, e a porta de acesso à ferramenta. O usuário permanece com o acesso à ferramenta original via <localhost:80> e acessa a nova ferramenta via <localhost:82>.

Antes de utilizar os comandos de construção da imagem e inicialização do contêiner, os arquivos contidos na pasta de referência passam também por modificações descritas na próxima subseção.

3.5.2 Pasta de referência "IQA Tool BRISQUE"

Uma cópia da pasta "IQA Tool DIQA" é criada e então renomeada como "IQA Tool BRISQUE". Nesta pasta, os arquivos "requirements.txt" e "app.ini" passam pelas edições descritas nesta subseção. À última linha dos requerimentos listados no arquivo "requirements.txt" é adicionado o "brisque", conforme listado abaixo, que passa a ser utilizado pela ferramenta para avaliar as imagens. A listagem completa deste arquivo encontra-se no Anexo B.

- IQA Tool BRISQUE/requirements.txt:

```
...
brisque
```

Conforme exposto abaixo, o arquivo "app.ini" passa por edição apenas na definição do "*socket*" para ficar compatível com a edição realizada no "docker-compose.yml".

- Versão do arquivo "app.ini" utilizado para iniciar a IQA Tool hospedando o DIQA:

```
[uwsgi]
module = run:app
socket = :8080
protocol = http
...
```

- Versão do "app.ini" utilizado para iniciar a IQA Tool, hospedando o BRISQUE, com alteração no socket:

```
[uwsgi]
module = run:app
socket = :8282
protocol = http
...
```

Ainda na pasta “IQA Tool BRISQUE”, além da edição do “requirements.txt” e do “app.ini”, é necessário modificar mais um arquivo para que a ferramenta então hospede o BRISQUE. Este arquivo é o “source.py”, localizado em “IQA Tool BRISQUE/app/”. Primeiramente, três novas bibliotecas devem ser importadas junto à listagem já existente neste código.

- Bibliotecas adicionadas ao arquivo “source.py”:

```
...
#Bibliotecas do BRISQUE
from brisque import BRISQUE
import numpy as np
from PIL import Image
...
```

Também a sua função “calculate_score” é modificada para o formato exibido abaixo. Algumas funções anteriormente utilizadas pelo DIQA podem ser retiradas deste código. O “source.py” em sua versão completa e limpa está no Anexo B.

- Função “calculate_score” modificada no arquivo “source.py”:

```
...
def calculate_score(imagens_folder_path):
    result_list = []
    image_list = glob.glob(imagens_folder_path + '/*.jpg')
    max_iteration = len(image_list)
    if max_iteration > 0:
        n = imagens_folder_path.replace(root_path, '')
        LOG(f'\nFolder: {n}, ({max_iteration} images) ')
        printProgressBar(0, max_iteration)
    for iteration, image_path in enumerate(image_list):
        #image = read_image(image_path) #diqa
        image = Image.open(image_path) #brisque
        ndarray = np.asarray(image) #brisque
        obj = BRISQUE(url=False)#brisque
        prediction = obj.score(img=ndarray) #brisque
        result = {
            'path': f'{image_path}',
            'prediction': prediction,
        }
```

```
            result_list.append(result)
            printProgressBar(iteration + 1, max_iteration)
        return result_list
...
```

Concluídas essas alterações, a ferramenta é construída, utilizando os comandos "docker compose build" e "docker compose up -d" no *Windows Power Shell* e acessada via <localhost:82> no navegador, enquanto a ferramenta que hospeda o DIQA continua disponível no <localhost:80>. Apesar de permanecer disponível no *layout* da segunda ferramenta, o *upload* de uma rede neural não tem relevância na execução da avaliação das imagens. Da mesma forma que no experimento anterior, o usuário deve fazer o *upload* da imagem em "Executions/New" para iniciar sua avaliação (ver Figura 13), e coletar a planilha de resultados em "Executions/Results" (Figura 14).

3.6 RESULTADOS

Em ambos os cenários, a avaliação é concluída com a entrega da planilha de resultados que fica disponível na aba de "Results" da IQA Tool. Os parâmetros utilizados para avaliar a qualidade das imagens diferem entre os métodos. A Tabela 4 exibe as notas obtidas pelos métodos, nas 20 imagens de resolução 512x384 pixels extraídas do HRIQ ilustradas na Tabela 3.

As pontuações obtidas pelo BRISQUE variam de 0 a 100, sendo os valores mais baixos indicadores de melhor qualidade de imagem (MITTAL; MOORTHY; BOVIK, 2012). As pontuações obtidas pelo DIQA variam de 0 a 1, onde a nota 1 aponta uma imagem de excelente qualidade (KIM; NGUYEN; LEE, 2019).

Enquanto o BRISQUE apresentou maior flutuação nas notas atribuídas às 20 imagens utilizadas neste experimento, o DIQA teve uma avaliação mais homogênea atribuída a todas as imagens, considerando todas de baixa qualidade. A sugestão de modificações nos algoritmos ou criação de um novo algoritmo foge ao escopo desta Dissertação.

Os resultados apresentados na subseção seguinte elucidam o tempo e os recursos requeridos pela ferramenta para avaliar diferentes quantidades e tamanhos de imagens utilizando o DIQA e o BRISQUE. Vale ressaltar que os métodos utilizados têm seu desempenho influenciado pelas dimensões das imagens a serem avaliadas, o que pode comprometer a confiabilidade das notas atribuidas pelos algoritmos às imagens submetidas.

3.6.1 Desempenho da IQA Tool

Nestes experimentos não foram utilizados orquestradores para limitar o uso de recursos por parte do Docker Desktop. A administração dos recursos da máquina e alocação de memória RAM ficaram livres e a cargo do sistema operacional hospedeiro, no caso, o

Tabela 4 – Notas do DIQA e do BRISQUE na avaliação das 20 imagens com dimensões 512x384 pixels usadas como teste.

Imagem	**DIQA**	**BRISQUE**
1101.jpg	0,2247	40,0887
1102.jpg	0,2270	7,7362
1103.jpg	0,2264	20,8558
1104.jpg	0,2265	13,5430
1105.jpg	0,2262	0,8651
1106.jpg	0,2275	5,9398
1107.jpg	0,2250	0,3893
1108.jpg	0,2272	26,5770
1109.jpg	0,2274	0,3096
1110.jpg	0,2274	5,8769
1111.jpg	0,2257	28,1906
1112.jpg	0,2272	14,9171
1113.jpg	0,2270	19,9351
1114.jpg	0,2281	15,3463
1115.jpg	0,2263	11,7764
1116.jpg	0,2255	6,4910
1117.jpg	0,5547	41,7844
1118.jpg	0,2245	45,6441
1119.jpg	0,2240	10,6703
1120.jpg	0,2270	9,0419

Fonte: Elaborado pela autora

Microsoft Windows 11. A Tabela 5 expõe os resultados obtidos pela ferramenta sob as condições nela descritas, em apenas uma execução em cada caso.

Todas as imagens de cenas naturais utilizadas foram extraídas do banco de dados HRIQ (HUANG QIANG WAN, 2024). Os arquivos executáveis do Docker referentes às duas versões da IQA Tool têm tamanhos semelhantes devido às sutis modificações realizadas entre elas. O tempo requerido para a construção desses arquivos também foi próximo para ambas as versões da ferramenta. A versão utilizando o DIQA mostrou-se bastante sensível às dimensões das imagens avaliadas, chegando a utilizar 100% da capacidade da CPU do sistema local ao avaliar 10 imagens com dimensões de 2.880×2.160 pixels. Em contraste, a ferramenta embarcada com o BRISQUE demandou valores baixos de memória RAM e CPU em todos os momentos. Em relação ao tempo de processamento, as duas versões da IQA Tool apresentaram comportamentos aproximados.

Tabela 5 – Resultados obtidos pelas ferramentas sob as condições distintas.

	DIQA	**BRISQUE**
Tempo para construção da imagem Docker	364 s	349 s
Tamanho da imagem Docker	4,04 GB	4,04 GB
Tempo total para avaliar 20 imagens 512x384p	5 s	4 s
Pico de utilização de memória RAM	4,85 GB	3,48 GB
Pico de utilização da capacidade da CPU	63,76%	20%
Tempo total para avaliar 1120 imagens 512x384p	252 s	220 s
Pico de utilização de memória RAM	4,86 GB	3,63 GB
Pico de utilização da capacidade da CPU	70%	20%
Tempo total para avaliar 70 imagens 1.024x768p	53 s	51 s
Pico de utilização de memória RAM	4,89 GB	3,76 GB
Pico de utilização da capacidade da CPU	90%	16%
Tempo total para avaliar 10 imagens 2.880x2.160p	74 s	64 s
Pico de utilização de memória RAM	4,90 GB	4,44 GB
Pico de utilização da capacidade da CPU	100%	15%

Fonte: Elaborado pela autora

3.6.2 Análise temporal de utilização do Docker

Esta subseção de experimentos e resultados analisa o tempo médio necessário para a IQA Tool avaliar 10 imagens (arquivos 1101.jpg a 1110.jpg do banco de dados HRIQ, previamente mostradas na Tabela 3 com dimensões de 2.880 × 2.160 pixels), comparando a execução da ferramenta em dois cenários: com o Docker Desktop e sem ele.

Para analisar a viabilidade de rodar a ferramenta sem o Docker, foi criado um ambiente virtual Linux no sistema Windows local, utilizando o WSL (*Windows Subsystem for Linux*). Nesse ambiente, os mesmos requisitos especificados no arquivo "requirements.txt" foram instalados, tornando-o apto a hospedar a aplicação Flask correspondente à IQA Tool. A criação do ambiente virtual e o uso do WSL permitiram a instalação de bibliotecas Linux, como o uWSGI, $tensorflow$, e outras dependências necessárias para a execução da ferramenta diretamente no laptop com Windows, sem a necessidade do Docker Desktop.

A Tabela 6 apresenta os resultados obtidos após 30 repetições dos testes em cada cenário. Vale destacar que, nesses experimentos, não foram utilizados orquestradores para limitar o uso de recursos tanto no Docker Desktop, quanto no ambiente virtual.

Os resultados apresentados na Tabela 6 indicam uma maior agilidade na avaliação das imagens ao hospedar a IQA Tool diretamente no laptop, por meio da criação de um ambiente virtual. Em outras palavras, o uso do Docker retardou a conclusão da tarefa no cenário avaliado, no caso do DIQA, em média, usou 37,8% mais tempo, e, no caso do BRISQUE, precisou de cerca de 13% mais tempo.

Tabela 6 – Tempo médio consumido para avaliação de 10 imagens 2.880 × 2.160 pixels com e sem a utilização do Docker Desktop.

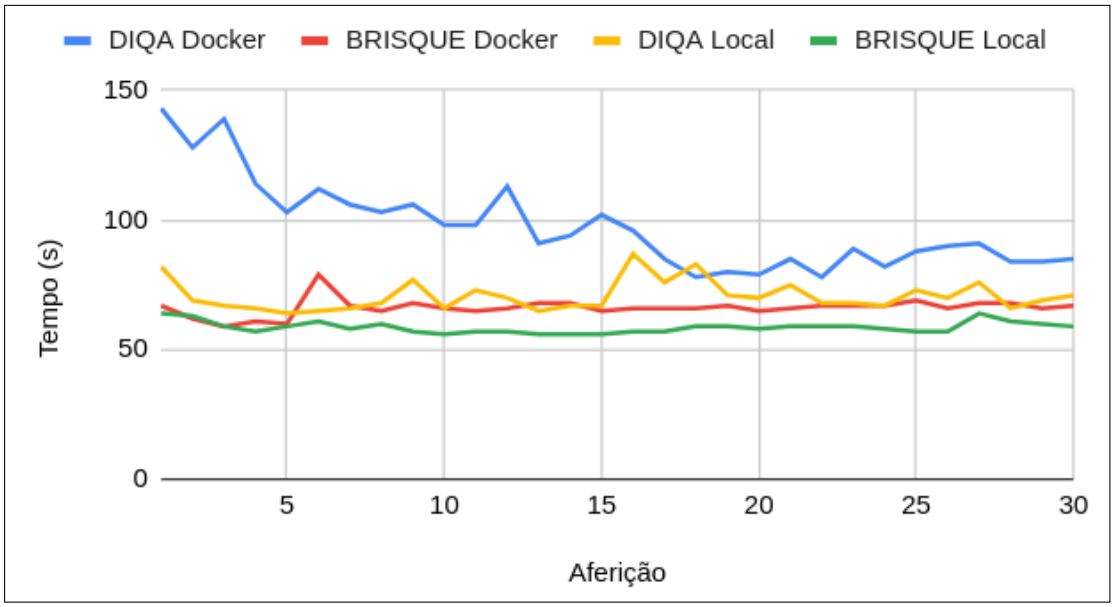
	Tempo médio	**Desvio padrão**
Tempo para avaliar as imagens com o Docker		
IQA Tool (DIQA)	97,47 s	17,09 s
IQA Tool (BRISQUE)	66,23 s	3,39 s
Tempo para avaliar as imagens sem o Docker		
IQA Tool (DIQA)	70,73 s	5,73 s
IQA Tool (BRISQUE)	58,63 s	2,22 s

Fonte: Elaborado pela Autora

De forma geral, os resultados mostram que o método de avaliação de imagens baseado em *Natural Scene Statistics*, o BRISQUE, foi mais rápido que o DIQA, que utiliza uma rede neural convolucional. A redução no tempo de execução do BRISQUE em comparação ao DIQA foi de 32% ao hospedar as ferramentas no Docker e de 17,11% ao executá-las diretamente no ambiente virtual.

O desvio padrão do tempo aferido ao utilizar a IQA Tool com o DIQA, quando hospedada no Docker, foi significativamente mais elevado em relação aos demais cenários. O gráfico na Figura 15 ilustra as 30 amostras obtidas em cada cenário. Nele, é possível observar que a ferramenta, ao utilizar o DIQA com o Docker, demandou mais tempo para realizar as avaliações iniciais, apresentando uma melhora ao longo do experimento. Nos outros três cenários, embora também tenham apresentado flutuações no tempo de execução, não foi observada uma tendência clara de aumento ou redução ao longo do experimento.

Figura 15 – Tempo de execução de 10 imagens 2880 × 2160 pixels. No gráfico, “Docker” ou “Local” refere-se ao uso do preditor com ou sem o Docker.



Fonte: Elaborado pela Autora.

4 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho foi bem-sucedido na criação da IQA Tool, uma ferramenta com interface *web* voltada a avaliação automática da qualidade de imagens sem referência NR-IQA fazendo uso apenas de recursos gratuitos. A ferramenta obtida requer etapas práticas para sua instalação, inicialização e utilização por meio de uma interface *web* amigável ao usuário.

A interface *web* foi desenvolvida utilizando o Flask, uma biblioteca gratuita e de código aberto para Python que facilita o desenvolvimento de aplicações *web* de maneira simples e flexível (DWYER, 2016), (GRINBERG, 2018). Para hospedar a IQA Tool em um computador Windows 11 de forma segura, rápida e prática, foi utilizado o Docker Desktop, ferramenta de criação de contêineres gratuita para o uso pessoal (DOCKER, 2024b), (STONEMAN, 2019).

Este trabalho consolidou o processo de *deploy* da IQA Tool em três etapas consistentes através das quais o usuário dispõe da IQA Tool em sua máquina local, lacalhost:80. São essas as três etapas:

- Instalação do Docker Desktop, realizado através da sua plataforma oficial <https://www.docker.com/products/docker-desktop/>;
- Aquisição dos arquivos que compõem a ferramenta disponibilizados via Github <https://github.com/MZMLPE/MSc-Tool>;
- Execução de dois comandos para construir a imagem e inicializar o contêiner Docker através do *Windows PowerShell*:

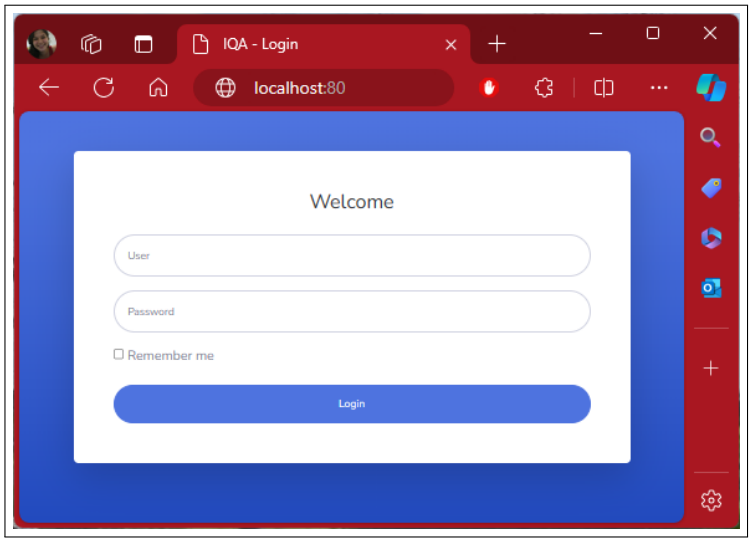
```
> docker compose build
> docker compose up -d
```

A Seção 4.1 trás considerações acerca da também bem sucedida implementação da IQA Tool no ambiente virtual realizada por fins experimentais. Comprovando ser um ambiente favorável alternativo ao uso do Docker Desktop.

A interface da IQA Tool obtida está exibida na Figura 16.

O primeiro método embarcado na IQA Tool foi o *Preditor de Avaliação de Qualidade de Imagem Profunda* (DIQA). As imagens inseridas na ferramenta são pré-processadas de acordo com essa metodologia, descrita pelo seu autor em Kim, Nguyen e Lee (2019). Após o pré-processamento, as imagens são avaliadas pelo modelo de previsão, a CNN treinada disponível nos arquivos da ferramenta sob o nome “model.h5”.

Figura 16 – Interface *web* inicial da IQA Tool para login do usuário.



Fonte: Elaborado pela autora.

O treinamento da CNN foi realizado utilizando imagens 500x500 pixels do *Live Dataset*, (SHEIKH et al., 2005). E as imagens submetidas à IQA Tool para avaliação foram selecionadas do banco de dados *High Resolution Image Quality* (HRIQ) e têm dimensões 512x384 pixels. Este banco de dados utilizado no Capítulo de Experimentos foi desenvolvido por Huang Qiang Wan (2024), e conta com 1.120 imagens de dimensões 2.880x2160 pixels, 1.024x768 pixels e 512x384 pixels.

Para as configurações de máquina utilizada neste trabalho, a ferramenta teve instalação rápida. A construção da imagem no Docker requeriu aproximadamente 364 segundos e a inicialização do contêiner, cerca de 3 segundos. Também o *upload* da pasta com 20 imagens a serem avaliadas pela ferramenta e a execução da avaliação foram rápidas, aproximadamente, 1 segundo e 11 segundos, respectivamente. Hospedando o DIQA ou o BRISQUE.

Aumentando a quantidade de imagens para 1.120, o tempo decorrido em sua avaliação é mais longo, aproximadamente, 374 segundos. Para esta quantidade de imagens, a IQA Tool não apresentou problema. No entanto, a inserção de imagens de alta resolução, ainda que seja uma pequena quantidade de imagens, 10 imagens 2.880x2.160 pixels, por exemplo, utiliza 100% da capacidade da CPU do sistema local.

Para adotar a capacidade de processamento de imagens de maior resolução, três opções podem ser consideradas: (i) ampliar as configurações de memória e capacidade de processamento da máquina hospedeira e dos recursos disponibilizados ao Docker Desktop; (ii) inserir uma etapa de recorte das imagens, onde a imagem inserida é processada e avaliada por partes; ou (iii) utilizar algoritmos rápidos para realizar o pré-processamento inicial.

Ao usuário Administrador da IQA Tool é possibilitada a edição do algoritmo de avali-

ação de imagens embarcado, bastando modificar o “model.h5” diretamente na aba “Configurações” da aplicação. Para isso é imprescindível verificar a compatibilidade do modelo com o original “.h5” embarcado, que une as informações do peso e da estrutura da CNN em um só arquivo. Essa estrutura pode variar conforme a versão utilizada da biblioteca Keras, que passou por atualizações recentes, (KERAS, 2023).

Quanto a modificar a ferramenta para hospedar o BRISQUE, vale ressaltar que foi necessária a modificação de poucos arquivos para adequar toda a aplicação a esse segundo modelo NR-IQA e que a capacidade de processamento do ambiente hospedeiro foi suficiente para dispor das duas versões da ferramenta operando de forma simultânea.

A personalização do algoritmo é motivada neste trabalho, no entanto, o usuário deve ficar atento ao acúmulo de resíduos no Docker. Caso haja lotação na memória alocada ao Docker, a ferramenta tem o desempenho afetado e apresenta erro. Para evitar esse cenário, este trabalho incentiva também a execução das rotinas de limpeza e remoção de resíduos, descrita em 2.2.1 para excluir contêineres e imagens em desuso e em 2.2.2 para excluir volumes.

Este trabalho limitou-se ao escopo de *deploy* da IQA Tool em um computador Windows 11, no entanto, a metodologia de instalação da ferramenta é a mesma a ser utilizada em um computador Linux, utilizando o Docker Engine ou Desktop, os arquivos disponibilizados no Github e as mesmas linhas de comando citadas anteriormente. A porta de acesso através do navegador também é a mesma. Localhost:80 para o DIQA e Localhost:82 para o BRISQUE.

4.1 DEPLOY SEM O DOCKER

Após a definição do arquivo “requirements.txt” com as bibliotecas e pacotes necessários para o funcionamento da ferramenta, a utilização de um ambiente virtual Linux mostrou-se uma alternativa viável ao uso do Docker para hospedar a IQA Tool.

No contexto deste trabalho, que se limita ao uso de um laptop com Windows 11 e configurações específicas, descritas previamente na Seção 3.1, o ambiente virtual demonstrou maior eficiência na hospedagem da IQA Tool, completando as avaliações de forma mais rápida. Embora a configuração do ambiente virtual tenha apresentado desafios, essa solução revelou-se mais eficaz do que a instalação via Docker.

4.2 AMEAÇAS À VALIDADE

Identificou-se que, ao hospedar a ferramenta no Docker, o tempo de avaliação foi superior, sugerindo um atraso possivelmente causado pelo Docker Desktop. Uma justificativa plausível é que o Docker, que utiliza o WSL, apresenta menor eficiência em comparação à instalação direta do WSL no sistema hospedeiro Windows. A Figura 4, apresentada an-

teriormente, indica uma maior praticidade pela eliminação da camada adicional associada ao Docker Engine.

Com base nos experimentos realizados, não foi possível estabelecer a superioridade do uso de ambientes virtuais em relação ao Docker para a avaliação de imagens de diferentes dimensões.

Adicionalmente, não se pode concluir que o uso de ambientes virtuais seja mais vantajoso do que o Docker Desktop em máquinas com sistema operacional Linux. Testes preliminares conduzidos em um computador com Linux indicam o contrário, demonstrando um desempenho superior da ferramenta ao utilizar o Docker, em vez da instalação direta dos requisitos em um ambiente virtual.

4.3 TRABALHOS FUTUROS

Como propostas futuras, este trabalho visualiza aprimorar o modelo de avaliação automática das imagens, principalmente, no quesito de resolução de imagens suportadas. O trabalho apresentado por SILVA (2022) propõe uma arquitetura do DIQA voltada para imagens de maior resolução, 12.000 $\times$ 9.000 pixels, com a qual obteve bons resultados na avaliação de imagens.

Também, parece promissora a utilização do banco de dados *High Resolution Image Quality* (HRIQ) no treinamento da rede neural embarcada. A ferramenta aqui desenvolvida se mostrou adequada para ser utilizada para testar o desempenho de algoritmos concorrentes.

Com a configuração atual, a ferramenta executa uma pasta por vez. O usuário deve aguardar a conclusão da avaliação para iniciar uma avaliação seguinte. Essa capacidade de armazenar demandas ou executá-las em paralelo é algo a ser desenvolvido em uma versão futura da IQA Tool.

Além disso, as configurações de compartilhamento de volumes entre contêineres distintos mostram-se atrativas. Com essa implementação, as duas versões da IQA Tool poderiam utilizar as mesmas pastas de imagens a serem avaliadas. As imagens não precisariam ser inseridas duas vezes, uma em cada ferramenta, otimizando o espaço ao compartilhar o mesmo repositório de imagens (DOCKER, 2024b).

Em relação aos testes de desempenho da ferramenta em diferentes configurações, planejamos expandir os cenários de teste e revisar a literatura relevante sobre o tema.

REFERÊNCIAS

AL., A. M. P. et. Performance evaluation of docker container and virtual machines. *Procedia Computer Science*, p. 1419–1428, 2020. Third International Conference on Computing and Network Communications (CoCoNet'19).

ALLEN, M. O. G. *The Definitive Guide to SQLite*. New York, NY: Apress, 2010.

ANDERSON, C. Docker [software engineering]. *IEEE Software*, v. 32, n. 3, p. 102–c3, 2015.

BERNSTEIN, D. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, v. 1, n. 3, p. 81–84, 2014.

BOBY, S.; SHARMIN, S. Medical image denoising techniques against hazardous noises: An iqa metrics based comparative analysis. *International Journal of Image, Graphics and Signal Processing*, v. 13, p. 25–43, 04 2021.

CHEN, S.; STROMER, D.; ALABDALRAHIM, H.; SCHWAB, S.; WEIH, M.; MAIER, A. Automatic dementia screening and scoring by applying deep learning on clock-drawing tests. *Scientific Reports*, v. 10, 11 2020.

CHOW, L. S.; PARAMESRAN, R. Review of medical image quality assessment. *Biomedical Signal Processing and Control*, v. 27, p. 145–154, 2016. ISSN 1746-8094. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1746809416300180>.

COMBE, T.; MARTIN, A.; PIETRO, R. D. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, v. 3, n. 5, p. 54–62, 2016.

DOCKER, I. *Docker Compose Quickstart*. [S.l.], 2024. Accessed: 2024-05-21. Disponível em: <https://docs.docker.com/compose/gettingstarted/>.

DOCKER, I. *Docker Documentation*. [S.l.], 2024. Accessed: 2024-05-21. Disponível em: <https://docs.docker.com/>.

DOCKER, I. *Install Docker Desktop on Windows*. [S.l.], 2024. Accessed: 2024-05-21. Disponível em: <https://docs.docker.com/desktop/install/windows-install/>.

DOCKER, I. *Persisting container data*. [S.l.], 2024. Accessed: 2024-05-21. Disponível em: <https://docs.docker.com/guides/docker-concepts/running-containers/persisting-container-data/>.

DOCKER, I. *Volumes*. [S.l.], 2024. Accessed: 2024-06-21. Disponível em: <https://docs.docker.com/storage/volumes/>.

DOCKER, I. *Volumes top-level element*. [S.l.], 2024. Accessed: 2024-05-21. Disponível em: <https://docs.docker.com/compose/compose-file/07-volumes/>.

DWYER, G. *Flask by Example*. Birmingham, UK: Packt publishing, 2016.

FANG, Y.; ZHU, H.; ZENG, Y.; MA, K.; WANG, Z. Perceptual quality assessment of smartphone photography. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2020. p. 3674–3683.

FONG, C.-M.; WANG, H.-W.; KUO, C.-H.; HSIEH, P.-C. Image quality assessment for advertising applications based on neural network. *Journal of Visual Communication and Image Representation*, v. 63, p. 102593, 2019. ISSN 1047-3203. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1047320319302147>.

GALBALLY, J.; MARCEL, S.; FIERREZ, J. Image quality assessment for fake biometric detection: Application to iris, fingerprint, and face recognition. *IEEE Transactions on Image Processing*, v. 23, n. 2, p. 710–724, 2014.

GRINBERG, M. *Flask Web Development*. Sebastopol, CA: O'Reilly Media, 2018.

HACHEM, C. E.; PERROT, G.; PAINVIN, L.; COUTURIER, R. Automation of quality control in the automotive industry using deep learning algorithms. In: *2021 International Conference on Computer, Control and Robotics (ICCCR)*. [S.l.: s.n.], 2021. p. 123–127.

HUANG QIANG WAN, J. K. H. High resolution image quality database. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, p. 3105–3109, 2024.

HUNT, J. Pip and conda virtual environments. In: *Advanced Guide to Python 3 Programming*. [S.l.]: Springer, 2023. p. 643–658.

HüTTEN, N.; GOMES, M. A.; HöLKEN, F.; ANDRICEVIC, K.; MEYES, R.; MEISEN, T. Deep learning for automated visual inspection in manufacturing and maintenance: A survey of open- access papers. *Applied System Innovation*, v. 7, p. 11, 01 2024.

JAIN, P.; TANEJA, K.; TANEJA, H. Advertisement image classification using deep learning with bert: A novel approach exploiting textual features. In: _____. [S.l.: s.n.], 2024. p. 443–456. ISBN 978-981-99-6546-5.

JORGENSON, D.; VU, K. The ict revolution, world economic growth, and policy issues. *Telecommunications Policy*, v. 40, 02 2016.

KERAS. Migrating keras 2 code to multi-backend keras 3. In: . [s.n.], 2023. Disponível em: <https://keras.io/guides/migrating_to_keras_3/>.

KIM, J.; LEE, S. Fully deep blind image quality predictor. *IEEE Journal of Selected Topics in Signal Processing*, v. 11, n. 1, p. 206–220, 2017.

KIM, J.; NGUYEN, A.-D.; LEE, S. Deep cnn-based blind image quality predictor. *IEEE transactions on neural networks and learning systems*, IEEE, v. 30, n. 1, p. 11–24, 2019.

KLIMA, M.; PATA, P.; FLIEGEL, K.; HANZLIK, P. Image quality evaluation in security imaging systems. *IEEE Aerospace and Electronic Systems Magazine*, v. 22, n. 1, p. 22–25, 2007.

KWON, S.; LEE, J. H. Divds: Docker image vulnerability diagnostic system. *IEEE Access*, p. 42666–42673, 2020.

LI, J.; YU, J.; XU, L.; XUE, X.; CHANG, C.-C.; MAO, X.; HU, J. A cascaded algorithm for image quality assessment and image denoising based on cnn for image security and authorization. *Security and Communication Networks*, n. 1, p. 8176984, 2018. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2018/8176984>.

MITTAL, A.; MOORTHY, A.; BOVIK, A. No-reference image quality assessment in the spatial domain. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, v. 21, p. 4695–4708, 08 2012.

MITTAL, A.; SOUNDARARAJAN, R.; BOVIK, A. C. Making a "completely blind" image quality analyzer. *IEEE Signal Processing Letters*, v. 20, n. 3, p. 209–212, 2013.

MOHAMMADI, P.; EBRAHIMI-MOGHADAM, A.; SHIRANI, S. Subjective and objective quality assessment of image: A survey. *Majlesi Journal of Electrical Engineering*, v. 9, 06 2014.

MOORTHY, A.; BOVIK, A. Blind image quality assessment: From natural scene statistics to perceptual quality. *Image Processing, IEEE Transactions on*, v. 20, p. 3350 – 3364, 01 2012.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *Icml*. [S.l.: s.n.], 2010.

OORT, B. V.; CRUZ, L.; ANICHE, M.; DEURSEN, A. V. The prevalence of code smells in machine learning projects. In: IEEE. *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. [S.l.], 2021. p. 1–8.

RAD HARRISON JOHN BHATTI, M. A. B. B. An introduction to docker and analysis of its performance. *IJCSNS International Journal of Computer Science and Network Security*, IJCSNS, v. 17, n. 3, p. 228–235, 2017.

REDI, J.; ZHU, Y.; RIDDER, H.; HEYNDERICKX, I. How passive image viewers became active multimedia users. In: _____. [S.l.: s.n.], 2015. p. 31–72. ISBN 978-3-319-10367-9.

SAAD, M. A.; BOVIK, A. C.; CHARRIER, C. Blind image quality assessment: A natural scene statistics approach in the dct domain. *IEEE Transactions on Image Processing*, v. 21, n. 8, p. 3339–3352, 2012.

SANTOKHI, M. *Active Learning in Image Quality Assessment*. Dissertação (Master Thesis) — Delft University of Technology, Netherlands, July 2017.

SHEIKH, H.; SABIR, M.; BOVIK, A. A statistical evaluation of recent full reference image quality assessment algorithms. *IEEE Transactions on Image Processing*, v. 15, n. 11, p. 3440–3451, 2006.

SHEIKH, H. R.; WANG, Z.; CORMACK, L.; BOVIK, A. C. Live image quality assessment database release 2 (2005). *URL http://live. ece. utexas. edu/research/quality*, 2005.

SILVA, S. J. D. A. Avaliação automática da qualidade de imagens de alta resolução sem referência. In: PROGRAMA DE PóS-GRADUAçãO EM CIêNCIAS DA COMPUTAçãO DA UNIVERSIDADE FEDERAL DE PERNAMBUCO. *Tese de Doutorado*. [S.l.], 2022. p. 27.

STONEMAN, E. Docker on windows: From 101 to production with docker on windows. In: PACKT, BIRMINGHAM - MUMBAI. *Docker on Windows second edition*. [S.l.], 2019. p. 9–11.

SUN, W.; ZHOU, F.; LIAO, Q. Mdid: A multiply distorted image database for image quality assessment. *Pattern Recognition*, Elsevier, v. 61, p. 153–168, 2017.

TANG, H.; JOSHI, N.; KAPOOR, A. Learning a blind measure of perceptual image quality. In: *CVPR 2011*. [S.l.: s.n.], 2011. p. 305–312.

VARGA, D. Empirical evaluation of full-reference image quality metrics on mdid database. *ArXiv*, abs/1910.01050, 2019. Disponível em: <https://api.semanticscholar.org/CorpusID:203626713>.

VO, A.; TRAN, H.; LE, T. Advertisement image classification using convolutional neural network. In: . [S.l.: s.n.], 2017. p. 197–202.

XU, S. S. P.; CHU, X. Performance evaluation of deep learning tools in docker containers. *3rd International Conference on Big Data Computing and Communications (BIGCOM)*, p. 395–403, d, 2017.

YE, P.; DOERMANN, D. No-reference image quality assessment using visual codebooks. *IEEE Transactions on Image Processing*, v. 21, n. 7, p. 3129–3138, 2012.

YU, X.; BAMPIS, C. G.; GUPTA, P.; BOVIK, A. C. Predicting the quality of images compressed after distortion in two steps. *IEEE Transactions on Image Processing*, v. 28, n. 12, p. 5757–5770, 2019.

YU, X.; BAMPIS, C. G.; GUPTA, P.; BOVIK, A. C. Predicting the quality of images compressed after distortion in two steps. *IEEE Transactions on Image Processing*, v. 28, n. 12, p. 5757–5770, 2019.

ŠPAčEK, F.; SOHLICH, R.; DULíK, T. Docker as platform for assignments evaluation. *Procedia Engineering*, v. 100, p. 1665–1671, 2015. ISSN 1877-7058. 25th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2014. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877705815005688>.

APÊNDICE A – COMPONENTES DA PASTA APP (DIQA)

Listing A.1 – “docker-compose.yml”

```
version: '3.7'
services:
  iqa_diqa:
    build: ./IQA Tool DIQA
    container_name: iqa_diqa_final
    restart: always
    ports:
      - '80:8080'
```

Listing A.2 – “app.ini”

```
[uwsgi]
module = run:app
socket = :8080
protocol = http
process = 4
threads = 4
master = true
chmod-socket = 660
vacuum = true
die-on-term = true
enable-threads = true
reload-on-rss = 4048
harakiri = 600
```

Listing A.3 – tasks.py

```

from fileinput import filename
import time
import xlsxwriter
from datetime import datetime
from .source import fuso_horario, main_source
from .log import WARNING, LOG


def threaded_task(app_context, folder, iqa_exec, db):
    tab_name = folder
    LOG(f'{iqa_exec}')
    app_context.push()
    result_name = '{}_{}_results_FV.xlsx'.format(folder, datetime.now().astimezone(
        fuso_horario).strftime('%d%m%Y_%H%M'))
    workbook = xlsxwriter.Workbook('app/results/{}'.format(result_name))
    folder_path = 'app/images/' + folder
    try:
        main_source(folder_path, workbook, tab_name)
        iqa_exec.progress = 100
        iqa_exec.result_filename = result_name
    except Exception as e:
        print(e)
```

```
23      workbook.close()
        if iqa_exec.progress != 100:
25          iqa_exec.progress = 0
        db.session.add(iqa_exec)
27      db.session.commit()
```

Listing A.4 – requirements.txt

```
1  Flask==2.2.2
   Flask-Login==0.6.2
3  Flask-SQLAlchemy==2.5.1
   h5py==3.7.0
5  imageio==2.21.1
   Jinja2==3.1.2
7  keras==2.9.0
   Keras-Preprocessing==1.1.2
9  numpy==1.23.1
   oauthlib==3.2.0
11 opencv-python==4.6.0.66
   Pillow==9.2.0
13 scikit-image==0.19.3
   scipy==1.9.0
15 SQLAlchemy==1.4.40
   XlsxWriter==3.0.3
```

Listing A.5 – source.py

```
   # @title Functions to detect issue
2  import numpy as np
   import cv2 as cv
4  import math

6  from skimage import feature
   import os.path
8  import tensorflow as tf
   import glob
10 import math
   import xlsxwriter
12 import os
   import time
14 from datetime import datetime, timezone, timedelta
   from tensorflow.keras.layers import Conv2D, Dense, GlobalAveragePooling2D
16 from .log import LOG

18 # Constantes
   root_path = 'app/images/'
20 model_path = 'app/model.h5'
   fuso_horario = timezone(timedelta(hours=-3))
22 SCORE_RUIM = 0.7

24 # title Functions to image evaluation

26 def normalize_kernel(kernel: tf.Tensor) -> tf.Tensor:
       return kernel / tf.reduce_sum(kernel)
28
   def gaussian_kernel2d(kernel_size: int, sigma: float, dtype=tf.float32) -> tf.Tensor
       :
```

```
    _range = tf.range(kernel_size)
    x, y = tf.meshgrid(_range, _range)
    constant = tf.cast(tf.round(kernel_size / 2), dtype=dtype)
    x = tf.cast(x, dtype=dtype) - constant
    y = tf.cast(y, dtype=dtype) - constant
    kernel = 1 / (2 * math.pi * sigma ** 2) * tf.exp(-(x ** 2 + y ** 2) / (2 * sigma
        ** 2))
    return normalize_kernel(kernel)

def gaussian_filter(image: tf.Tensor, kernel_size: int, sigma: float, dtype=tf.
    float32) -> tf.Tensor:
    kernel = gaussian_kernel2d(kernel_size, sigma)
    if image.get_shape().ndims == 3:
        image = image[tf.newaxis, :, :, :]
    image = tf.cast(image, tf.float32)
    image = tf.nn.conv2d(image, kernel[:, :, tf.newaxis, tf.newaxis], strides=1,
        padding='SAME')
    return tf.cast(image, dtype)

def image_shape(image: tf.Tensor, dtype=tf.int32) -> tf.Tensor:
    shape = tf.shape(image)
    shape = shape[:2] if image.get_shape().ndims == 3 else shape[1:3]
    return tf.cast(shape, dtype)

def scale_shape(image: tf.Tensor, scale: float) -> tf.Tensor:
    shape = image_shape(image, tf.float32)
    shape = tf.math.ceil(shape * scale)
    return tf.cast(shape, tf.int32)

def rescale(image: tf.Tensor, scale: float, dtype=tf.float32, **kwargs) -> tf.Tensor
    :
    assert image.get_shape().ndims in (3, 4), 'The tensor must be of dimension 3 or
        4'
    image = tf.cast(image, tf.float32)
    rescale_size = scale_shape(image, scale)
    rescaled_image = tf.image.resize(image, size=rescale_size, **kwargs)
    return tf.cast(rescaled_image, dtype)

def image_preprocess(image: tf.Tensor) -> tf.Tensor:
    image = tf.cast(image, tf.float32) / 255.0
    image = tf.image.rgb_to_grayscale(image)
    image_low = gaussian_filter(image, 16, 7 / 6)
    image_low = rescale(image_low, 1 / 4, method=tf.image.ResizeMethod.BICUBIC)
    image_low = tf.image.resize(image_low, size=image_shape(image), method=tf.image.
        ResizeMethod.BICUBIC)
    return image - tf.cast(image_low, image.dtype)

def read_image(filename: str, **kwargs) -> tf.Tensor:
    stream = tf.io.read_file(filename)
    return tf.image.decode_image(stream, **kwargs)

def printProgressBar(iteration, total, prefix='Progress:', suffix='Complete',
    decimals=1, length=50, fill='',
                     printEnd=""):
    percent = ("{0:." + str(decimals) + "f}").format(100 * (iteration / float(total)
        ))
    filledLength = int(length * iteration // total)
```

```
        bar = fill * filledLength + '-' * (length - filledLength)
80      LOG(f'\r{prefix} |{bar}| {percent}% {suffix}')
        #print(f'\r{prefix} |{bar}| {percent}% {suffix}', end=printEnd)
82      # Print New Line on Complete
        if iteration == total:
84          LOG(' ')

86  def calculate_score(imagens_folder_path):
        model = tf.keras.models.load_model(model_path)
88
        result_list = []
90      image_list = glob.glob(imagens_folder_path + '/*.jpg')
        max_iteration = len(image_list)
92
        if max_iteration > 0:
94          n = imagens_folder_path.replace(root_path, '')
            LOG(f'\nFolder: {n}, ({max_iteration} images) ')
96          printProgressBar(0, max_iteration)

98      for iteration, image_path in enumerate(image_list):
            image = read_image(image_path)
100         prediction = model.predict(image_preprocess(image))[0][0]
            prediction = 1 if prediction > 1 else prediction
102         result = {
                'path': f'{image_path}',
104             'prediction': prediction,
            }
106         result_list.append(result)
            printProgressBar(iteration + 1, max_iteration)
108     return result_list

110 def write_results(worksheet, result_folder, index, format_default, folder_name):
        max_width0 = 5
112     max_width1 = 6
        max_width2 = 4
114
        max_iteration = 1
116     iteration = 0
        LOG(f'\nFixing score and writing results: {max_iteration} folders')  # muito
            lento
118     printProgressBar(iteration, max_iteration)

120     i = 0
        while i < len(result_folder):
122         filename_1 = result_folder[i].get('path').split(os.path.sep)[-1]
            max_width0 = max(len(filename_1), max_width0)
124         max_width2 = max(len(folder_name), max_width2)

126         result1 = result_folder[i].get('prediction')

128         col_index = 0
            worksheet.write(index, col_index, filename_1)
130         col_index += 1
            worksheet.write(index, col_index, folder_name)
132         col_index += 1
            worksheet.write(index, col_index, result1,
134                         format_default)
```

```
            col_index += 1
            index += 1
            i += 1

        return max_width0, max_width1, max_width2

    def auto_fit(worksheet, width0, width1, width2):
        # image, folder, type
        worksheet.set_column(0, 0, width0 + 5)
        worksheet.set_column(1, 1, width1 + 5)
        worksheet.set_column(2, 2, width2 + 5)
        # scores
        worksheet.set_column(3, 3, 11)
        # result, issue
        worksheet.set_column(4, 4, 6)
        worksheet.set_column(5, 5, 38)

#       worksheet.set_column('J:J', None, None, {'hidden': True})

    def main_source(folder_path, workbook, tab_name):
        LOG(f'\n...Starting device...\n')
        start_time = time.time()
        result = {}

        # workbook cell formats
        format_default = workbook.add_format({'num_format': '#,##0.0000'})

        result = calculate_score(folder_path)
        result.sort(key=lambda x: (x.get('path')))

        folder_name = folder_path.split('/')[-1].upper()

        worksheet = workbook.add_worksheet('{}'.format(tab_name))
        index = 0
        worksheet.write(index, 0, 'Image')
        worksheet.write(index, 1, 'Score')
        index += 1
        w0, w1, w2 = write_results(worksheet, result, index, format_default, folder_name
            )
        auto_fit(worksheet, w0, w1, w2)

        end_time = time.time()
        t = int(end_time - start_time)
        LOG(f'Tab {tab_name} added, duration {t} seconds')
        return result
```

Listing A.6 – rclone_cron.sh

```
#!/bin/bash
dupe_script=$(ps -ef | grep "rclone-cron.sh" | grep -v grep | wc -l)

if [ ${dupe_script} -gt 2 ]; then
    echo -e "rclone sync script was already running."
    exit 0
fi
    echo -e "rclone sync start"
    date
```

```
    rclone sync remote:BRRES64_Data/Images /home/mzml/images/
    #rclone sync drive_iqa:BRRES64_Data/Images /home/pesquisa/images/
    docker cp /home/mzml/images/. 21eb83c42996:/app/images
    date
    echo -e "rclone sync end"
exit
```

Listing A.7 – models.py

```
from flask_login import UserMixin
from . import db

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True) # primary keys are required by SQLAlchemy
    email = db.Column(db.String(100), unique=True)
    password = db.Column(db.String(100))
    name = db.Column(db.String(1000))
    admin = db.Column(db.Boolean, default=False)

class Execution(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    folder = db.Column(db.String(120))
    date = db.Column(db.DateTime)
    progress = db.Column(db.Integer, default=0)
    result_filename = db.Column(db.String(200))
```

Listing A.8 – main.py

```
import shutil
from datetime import datetime
from flask import Blueprint, render_template, redirect, url_for, request, send_file, current_app
from flask_login import login_required, current_user
import os
import zipfile
from threading import Thread
from .tasks import threaded_task
from .models import Execution
from .source import fuso_horario
from .log import LOG

from app import db

main = Blueprint('main', __name__)

BASE_DIR = os.path.dirname(os.path.abspath(__file__))


@main.route('/')
def login():
    if current_user.is_authenticated:
        return redirect(url_for('main.index'))
    return render_template('login.html')


@main.route('/index')
```

```
28 @login_required
   def index():
30     return render_template('index.html', usuario=current_user)

32
   @main.route('/config')
34 @login_required
   def config():
36     if not current_user.admin:
           return redirect(url_for('main.index'))
38     return render_template('config.html', usuario=current_user)

40
   @main.route('/execs')
42 @login_required
   def execs():
44     all_execs = Execution.query.order_by(Execution.id.desc()).all()
       return render_template('execs.html', usuario=current_user, executions=all_execs)
46

48 def get_folders():
       folders = []
50     for x in os.listdir('app/images'):
           if os.path.isdir('app/images/' + x):
52             folders.append(x)
       return folders
54

56 @main.route('/new_exec')
   @login_required
58 def new_exec_get():
       folders = get_folders()
60     return render_template('new_exec.html', usuario=current_user, folders=folders)

62
   @main.route('/new_exec', methods=['POST'])
64 @login_required
   def new_exec():
66     folder = request.form.get('folder')
       iqa_exec = Execution(folder=folder,
68                           date=datetime.now().astimezone(fuso_horario),
                             progress=50,
70                           result_filename=''
                             )
72     db.session.add(iqa_exec)
       db.session.commit()
74     print(iqa_exec, iqa_exec.progress)
       LOG(f'\nA\n')
76     thread = Thread(target=threaded_task, args=(current_app.app_context(), folder,
           iqa_exec, db))
       thread.daemon = True
78     thread.start()
       LOG(f'\nB\n')
80     return redirect(url_for('main.execs'))

82     return redirect(url_for('main.new_exec_get'))
```

```
84
@main.route('/delete_exec/<exec_id>')
86 @login_required
def delete_exec(exec_id):
88     execution = Execution.query.get(exec_id)
    if current_user.admin or (execution.progress == 100):
90         db.session.delete(execution)
        try:
92             if execution.result_filename != '':
                os.remove('app/results/' + execution.result_filename)
94         except Exception as e:
            print(e)
96         db.session.commit()
    return redirect(url_for('main.execs'))
98

100 @main.route('/upload', methods=['POST'])
@login_required
102 def upload():
    file = request.files.get('file')
104     if file and (file.filename.split('.')[-1] == 'h5' or file.filename.split('.')
        [-1] == 'tf'):
        file.save('app/model.h5')
106     return redirect(url_for('main.config'))

108
@main.route('/download/<path:filename>', methods=['GET'])
110 @login_required
def download_file(filename):
112     path = "results/" + filename
    return send_file(path, as_attachment=True)
114

116 def unzip_file(zip_src, dst_dir):
    r = zipfile.is_zipfile(zip_src)
118     if r:
        fz = zipfile.ZipFile(zip_src, "r")
120         for file in fz.namelist():
            fz.extract(file, dst_dir)
122     else:
        return "Please upload a compressed file of zip type"
124

126 def get_used_disk_space():
    t, u, f = shutil.disk_usage("/")
128     used = int((u / t) * 100)
    total = t // (2**30)
130     return used, total

132
@main.route("/upload_images", methods=["GET", "POST"])
134 def upload_images():
    if request.method == "GET":
136         folders = get_folders()
        used, total = get_used_disk_space()
138         return render_template("upload_images.html", usuario=current_user, folders=
            folders, used=used, total=total)
```

```
    obj = request.files.get("file")
    ret_list = obj.filename.rsplit(".", maxsplit=1)
    if len(ret_list) != 2:
        return "Please upload a compressed file of zip type"
    if ret_list[1] != "zip":
        return "Please upload a compressed file of zip type"

    file_path = os.path.join(BASE_DIR, "images", obj.filename)
    obj.save(file_path)
    target_path = os.path.join(BASE_DIR, "images")
    ret = unzip_file(file_path, target_path)
    os.remove(file_path)
    return redirect(url_for('main.upload_images'))


@main.route('/delete_folder/<folder>')
@login_required
def delete_folder(folder):
    if current_user.admin:
        shutil.rmtree('app/images/' + folder)
    return redirect(url_for('main.upload_images'))
```

Listing A.9 – log.py

```
import logging

logging.basicConfig(format='%(asctime)s %(levelname)s %(message)s')
logger = logging.getLogger()
logger.setLevel(logging.INFO)

LOG = logger.info
WARNING = logger.warning
```

Listing A.10 – auth.py

```
# auth.py

from flask import Blueprint, render_template, redirect, url_for, request, flash
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import login_user, logout_user, login_required, current_user
from .models import User
from . import db

auth = Blueprint('auth', __name__)


@auth.route('/login')
def login():
    return render_template('login.html')


@auth.route('/login', methods=['POST'])
def login_post():
    email = request.form.get('email')
    password = request.form.get('password')
    remember = True if request.form.get('remember') else False
    user = User.query.filter_by(email=email).first()
```

```
    # check if user actually exists
    # take the user supplied password, hash it, and compare it to the hashed password in database
    if not user or not check_password_hash(user.password, password):
        flash('Please check your login details and try again.')
        return redirect(url_for('auth.login'))  # if user doesn't exist or password is wrong, reload the page

    # if the above check passes, then we know the user has the right credentials
    login_user(user, remember=remember)
    return redirect(url_for('main.index'))


@auth.route('/users')
@login_required
def users():
    if current_user.admin:
        users = User.query.order_by(User.id.asc()).all()
        return render_template('users.html', usuario=current_user, users=users)
    else:
        return redirect(url_for('main.index'))


@auth.route('/user_edit/<user_id>', methods=['GET'])
@login_required
def user_edit_get(user_id):
    if current_user.admin:
        usuario = User.query.get(user_id)
        return render_template('edit_user.html', usuario=current_user, edit_user=usuario)
    else:
        return render_template('edit_user.html', usuario=current_user, edit_user=current_user)


@auth.route('/user_edit/<user_id>', methods=['POST'])
@login_required
def user_edit_post(user_id):

    name = request.form.get('name')
    password = request.form.get('password')
    password2 = request.form.get('password2')
    is_admin = True if request.form.get('isAdmin') == 'on' else False

    user = User.query.get(user_id)

    if password != password2:
        flash('Passwords must be the same')
        return redirect(url_for('auth.user_edit_get', user_id=user_id))

    if current_user.admin:
        user.name = name
        if password != '':
            user.password = generate_password_hash(password, method='sha256')
        user.admin = is_admin
        db.session.add(user)
```

```
76          db.session.commit()
        elif current_user.email == user.email:
78          user.name = name
            user.password = generate_password_hash(password, method='sha256')
80          db.session.add(user)
            db.session.commit()
82
        return redirect(url_for('auth.users'))
84

86  @auth.route('/user_delete/<user_id>')
    @login_required
88  def delete_user(user_id):
        if current_user.admin:
90          user = User.query.get(user_id)
            db.session.delete(user)
92          db.session.commit()
        return redirect(url_for('auth.users'))
94

96  @auth.route('/signup')
    @login_required
98  def signup():
        if current_user.admin:
100         return render_template('signup.html', usuario=current_user)
        else:
102         return redirect(url_for('main.index'))

104
    @auth.route('/signup', methods=['POST'])
106 @login_required
    def signup_post():
108     if not current_user.admin:
            return redirect(url_for('main.index'))
110
        email = request.form.get('email')
112     name = request.form.get('name')
        password = request.form.get('password')
114     is_admin = True if request.form.get('isAdmin') == 'on' else False

116     user = User.query.filter_by(email=email).first()

118     if user:
            flash('Email address already exists')
120         return redirect(url_for('auth.signup'))

122     # create new user with the form data. Hash the password so plaintext version isn
            't saved.
        new_user = User(email=email,
124                     name=name,
                        password=generate_password_hash(password, method='sha256'),
126                     admin=is_admin)

128     # add the new user to the database
        db.session.add(new_user)
130     db.session.commit()
```

```
132     return redirect(url_for('main.index'))

134
@auth.route('/logout')
136 @login_required
def logout():
138     logout_user()
        return redirect(url_for('auth.login'))
```

Listing A.11 – ___init___

```
1  from flask import Flask
   from flask_sqlalchemy import SQLAlchemy
3  from flask_login import LoginManager

5  # init SQLAlchemy so we can use it later in our models
   db = SQLAlchemy()
7
   def create_app():
9      app = Flask(__name__, static_url_path='',
               static_folder='static',
11             template_folder='templates')

13     app.config['SECRET_KEY'] = 'genericKeyIQAtool'
       app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///sqlite/db.sqlite'
15
       db.init_app(app)
17
       login_manager = LoginManager()
19     login_manager.login_view = 'auth.login'
       login_manager.init_app(app)
21
       from .models import User
23
       @login_manager.user_loader
25     def load_user(user_id):
           # since the user_id is just the primary key of our user table, use it in the
               query for the user
27         return User.query.get(int(user_id))

29     # blueprint for auth routes in our app
       from .auth import auth as auth_blueprint
31     app.register_blueprint(auth_blueprint)

33     # blueprint for non-auth parts of app
       from .main import main as main_blueprint
35     app.register_blueprint(main_blueprint)

37     return app
```

[templates:] pasta com os arquivos .HTML, utilizados para definição da aparência da Ferramenta e interação dos usuários. [static:] figuras auxiliares pra exibição na interface web. Por exemplo, foto genérica de usuário. [sqlite:] pasta com arquivo “db.sqlite”, banco de dados da ferramenta. [results:] pasta com as planilhas de resultados obtidos pela Ferramenta. Esta pasta é aditada pelo usuário. [images:] pasta com as imagens inseridas

da Ferramenta. [___pycache___:] cache com arquivos compilados de códigos python no formato .cpython-38.

APÊNDICE B – COMPONENTES MODIFICADOS PARA HOSPEDAR O BRISQUE

Listing B.1 – “docker-compose.yml”

```
version: '3.7'
services:
  iqa_brisque:
    build: ./IQA Tool BRISQUE
    container_name: iqa_brisque_final
    restart: always
    ports:
      - '82:8282'
```

Listing B.2 – “requirements.txt”

```
        absl-py==1.2.0\\
        astunparse==1.6.3\\
        cachetools==5.2.0\\
        certifi==2022.6.15\\
        charset-normalizer==2.1.0\\
        click==8.1.3\\
        Flask==2.2.2\\
        Flask-Login==0.6.2\\
        Flask-SQLAlchemy==2.5.1\\
        flatbuffers==1.12\\
        gast==0.4.0\\
        google-auth==2.10.0\\
        google-auth-oauthlib==0.4.6\\
        google-pasta==0.2.0\\
        greenlet==1.1.2\\
        grpcio==1.47.0\\
        h5py==3.7.0\\
        idna==3.3\\
        imageio==2.21.1\\
        importlib-metadata==4.12.0\\
        itsdangerous==2.1.2\\
        Jinja2==3.1.2\\
        keras==2.9.0\\
        Keras-Preprocessing==1.1.2\\
        libclang==14.0.6\\
        Markdown==3.4.1\\
        MarkupSafe==2.1.1\\
        networkx==2.8.5\\
        numpy==1.23.1\\
        oauthlib==3.2.0\\
        opencv-python==4.6.0.66\\
        opt-einsum==3.3.0\\
        packaging==21.3\\
        Pillow==9.2.0\\
        protobuf==3.19.4\\
        pyasn1==0.4.8\\
        pyasn1-modules==0.2.8\\
        pyparsing==3.0.9\\
```

```
        PyWavelets==1.3.0\\
        requests==2.28.1\\
        requests-oauthlib==1.3.1\\
        rsa==4.9\\
        scikit-image==0.19.3\\
        scipy==1.9.0\\
        six==1.16.0\\
        SQLAlchemy==1.4.40\\
        tensorboard==2.9.1\\
        tensorboard-data-server==0.6.1\\
        tensorboard-plugin-wit==1.8.1\\
        tensorflow==2.9.1\\
        tensorflow-estimator==2.9.0\\
        tensorflow-io-gcs-filesystem==0.26.0\\
        termcolor==1.1.0\\
        tifffile==2022.8.8\\
        typing\_extensions==4.3.0\\
        urllib3==1.26.11\\
        uWSGI==2.0.20\\
        Werkzeug==2.2.2\\
        wrapt==1.14.1\\
        XlsxWriter==3.0.3\\
        zipp==3.8.1\\
        brisque\\
```

Listing B.3 – “app.ini”

```
[uwsgi]
module = run:app
socket = :8282
protocol = http
process = 4
threads = 4
master = true
chmod-socket = 660
vacuum = true
die-on-term = true
enable-threads = true
reload-on-rss = 4048
harakiri = 600
```

Listing B.4 – source.py

```
# @title Functions to detect issue
import numpy as np
import cv2 as cv
import math

from skimage import feature
import os.path
import tensorflow as tf
import glob
import math
import xlsxwriter
import os
import time
from datetime import datetime, timezone, timedelta
```

```
15 from tensorflow.keras.layers import Conv2D, Dense, GlobalAveragePooling2D
   from .log import LOG
17
   #BibLiotecas do BRISQUE
19 from brisque import BRISQUE
   import numpy as np
21 from PIL import Image

23
   # Constantes
25 root_path = 'app/images/'
   model_path = 'app/model.h5'
27 fuso_horario = timezone(timedelta(hours=-3))
   SCORE_RUIM = 0.7
29

31 def printProgressBar(iteration, total, prefix='Progress:', suffix='Complete',
       decimals=1, length=50, fill='   ',
                       printEnd=""):
33     percent = ("{0:." + str(decimals) + "f}").format(100 * (iteration / float(total)
           ))
       filledLength = int(length * iteration // total)
35     bar = fill * filledLength + '-' * (length - filledLength)
       LOG(f'\r{prefix} |{bar}| {percent}% {suffix}')
37     #print(f'\r{prefix} |{bar}| {percent}% {suffix}', end=printEnd)
       # Print New Line on Complete
39     if iteration == total:
           LOG(' ')
41

43 def calculate_score(imagens_folder_path):
       #model = tf.keras.models.load_model(model_path) #DIQA
45
       result_list = []
47     image_list = glob.glob(imagens_folder_path + '/*.jpg')
       max_iteration = len(image_list)
49
       if max_iteration > 0:
51         n = imagens_folder_path.replace(root_path, '')
           LOG(f'\nFolder: {n}, ({max_iteration} images) ')
53         printProgressBar(0, max_iteration)

55     for iteration, image_path in enumerate(image_list):
           image = Image.open(image_path) #BRISQUE
57         ndarray = np.asarray(image) #BRISQUE
           obj = BRISQUE(url=False) #BRISQUE
59         prediction = obj.score(img=ndarray) #BRISQUE
           result = {
61             'path': f'{image_path}',
               'prediction': prediction,
63         }
           result_list.append(result)
65         printProgressBar(iteration + 1, max_iteration)
       return result_list
67

69 def write_results(worksheet, result_folder, index, format_default, folder_name):
```

```
    max_width0 = 5
    max_width1 = 6
    max_width2 = 4

    max_iteration = 1
    iteration = 0
    LOG(f'\nFixing score and writing results: {max_iteration} folders')  # muito lento
    printProgressBar(iteration, max_iteration)

    i = 0
    while i < len(result_folder):
        filename_1 = result_folder[i].get('path').split(os.path.sep)[-1]
        max_width0 = max(len(filename_1), max_width0)
        max_width2 = max(len(folder_name), max_width2)

        result1 = result_folder[i].get('prediction')

        col_index = 0
        worksheet.write(index, col_index, filename_1)
        col_index += 1
        worksheet.write(index, col_index, folder_name)
        col_index += 1
        worksheet.write(index, col_index, result1,
                        format_default)
        col_index += 1
        index += 1
        i += 1

    return max_width0, max_width1, max_width2


def auto_fit(worksheet, width0, width1, width2):
    # image, folder, type
    worksheet.set_column(0, 0, width0 + 5)
    worksheet.set_column(1, 1, width1 + 5)
    worksheet.set_column(2, 2, width2 + 5)
    # scores
    worksheet.set_column(3, 3, 11)
    # result, issue
    worksheet.set_column(4, 4, 6)
    worksheet.set_column(5, 5, 38)


#   worksheet.set_column('J:J', None, None, {'hidden': True})

def main_source(folder_path, workbook, tab_name):
    LOG(f'\n...Starting device...\n')
    start_time = time.time()
    result = {}

    # workbook cell formats
    format_default = workbook.add_format({'num_format': '#,##0.0000'})

    result = calculate_score(folder_path)
    result.sort(key=lambda x: (x.get('path')))

```

```
        folder_name = folder_path.split('/')[-1].upper()
127
        worksheet = workbook.add_worksheet('{}'.format(tab_name))
129     index = 0
        worksheet.write(index, 0, 'Image')
131     worksheet.write(index, 1, 'Score')
        index += 1
133     w0, w1, w2 = write_results(worksheet, result, index, format_default, folder_name
            )
        auto_fit(worksheet, w0, w1, w2)
135
        end_time = time.time()
137     t = int(end_time - start_time)
        LOG(f'Tab {tab_name} added, duration {t} seconds')
139     return result
```