UNIVERSIDADE FEDERAL DO CEARÁ

INSTITUTO UNIVERSIDADE VIRTUAL

CURSO DE GRADUAÇÃO EM SISTEMAS E MÍDIAS DIGITAIS

LUCAS DE LIMA SILVA

EXPERIMENTOS DE TRANSFERÊNCIA NEURAL DE ESTILO EM APLICAÇÕES DE VISÃO COMPUTACIONAL

FORTALEZA

2023

LUCAS DE LIMA SILVA

EXPERIMENTOS DE TRANSFERÊNCIA NEURAL DE ESTILO EM APLICAÇÕES DE VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas e Mídias Digitais do Instituto Universidade Virtual da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas e Mídias Digitais.

Orientador: Prof. Dr. Rafael Augusto Ferreira do Carmo.

FORTALEZA

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S581e Silva, Lucas de Lima.
Experimentos de transferência neural de estilo em aplicações de visão computacional / Lucas de Lima Silva. – 2023.
57 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Instituto UFC Virtual, Curso de Sistemas e Mídias Digitais, Fortaleza, 2023.
Orientação: Prof. Dr. Rafael Augusto Ferreira do Carmo.

1. Visão computacional. 2. Python. 3. Detecção de objeto. 4. Transferência neural de estilo. 5. Redes neurais convolucionais. I. Título.

CDD 302.23

LUCAS DE LIMA SILVA

EXPERIMENTOS DE TRANSFERÊNCIA NEURAL DE ESTILO EM APLICAÇÕES DE VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas e Mídias Digitais do Instituto Universidade Virtual da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas e Mídias Digitais.

Aprovada em: 11/07/2023.

BANCA EXAMINADORA

Prof. Dr. Rafael Augusto Ferreira do Carmo (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Adriano Anunciação Oliveira
Universidade Federal do Ceará (UFC)

Prof. Dr. Antônio José Melo Leite Júnior
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Ao Prof. Dr. Rafael Augusto Ferreira do Carmo, pela excelente orientação e ajuda na execução e idealização desse trabalho.

Aos meus amados pais, Maria Leuda de Lima Silva e Isaías Ferreira da Silva, que embora não tenham tido a oportunidade de receber uma educação básica formal, eles trabalharam arduamente para proporcionar a mim, seu filho, a oportunidade de obter uma educação de nível superior.

À minha querida irmã, Irislene de Lima Silva, cujo apoio e exemplo são uma fonte constante de inspiração e força.

Aos colegas de turma, em especial aos membros do grupo Gartic, que foram uma válvula de escape e de extrema ajuda no decorrer do curso.

À Universidade Federal do Ceará e seus servidores, em particular aos docentes do curso de Sistemas e Mídias Digitas, que além de instigar na busca de conhecimento sempre demonstraram um cuidado genuíno e empatia.

RESUMO

O trabalho em questão visa propor uma arquitetura por intermédio da elaboração e prototipação de soluções em realidade aumentada que combinam detecção de objeto e transferência neural de estilo. Ao longo do trabalho, serão exploradas tecnologias para lidar com os desafios mencionados, além de apontar as limitações enfrentadas no desenvolvimento, como o desempenho computacional, a complexidade dos modelos e a otimização dos resultados. Foi necessária uma pesquisa prévia para levantamento de tecnologias para detecção de objeto, transferência neural de estilo e realidade aumentada. Para a elaboração e prototipação do trabalho foram utilizadas as tecnologias ARCore, YOLO, redes neurais convolucionais, Python, Pytorch e a biblioteca OpenCV. Este trabalho apresenta o desenvolvimento de três protótipos, sendo que o terceiro é disponibilizado em duas versões distintas: a original e uma versão aprimorada com a implementação de melhorias significativas de desempenho. Como resultado dos protótipos, destaca-se no primeiro a utilização de detecção de objeto e a manipulação do objeto detectado, contudo, no segundo destaca-se a integração entre a detecção de objeto e a transferência neural de estilo somente nos objetos detectados, mantendo todo resto da cena em sua forma original, ademais como terceiro e último protótipo manteve-se a funcionalidade do segundo protótipo, porém agora cada classe dos objetos detectados terá um dos dois estilos escolhidos para o trabalho, que no caso foram as obras "*Self-portrait*" de Pablo Picasso e “O Gato” de Romero Britto. Por fim, foi considerado o desempenho das soluções desenvolvidas em certos contextos, para ser possível verificar a eficácia e a adaptabilidade delas, mas podemos destacar a capacidade da solução rodar a mais de 60 quadros por segundo, no primeiro protótipo e cerca de 10 quadros no terceiro protótipo, utilizando uma máquina com placa gráfica RTX 2060 para executar a solução em ambos os casos citados.

Palavras-chave: Visão computacional. Python. Detecção de objetos. Transferência neural de estilo. Aprendizado de máquina. Redes neurais convolucionais.

ABSTRACT

The objective of this work is the development and prototyping of augmented reality solutions that combine object detection and neural style transfer. Throughout the work, technologies will be explored to address the mentioned challenges, in addition to highlighting the limitations faced during development, such as computational performance, model complexity, and result optimization. A preliminary research was required to survey technologies for object detection, neural style transfer, and augmented reality. For the elaboration and prototyping of the work, technologies such as ARCore, YOLO, convolutional neural networks, Python, PyTorch, and the OpenCV library were utilized. This study presents the development of three prototypes, with the third being provided in two distinct versions: the original and an enhanced version with the implementation of significant performance improvements. As a result of the prototypes, the first one emphasizes object detection and manipulation of the detected object, while the second highlights the integration of object detection and neural style transfer exclusively on the detected objects, preserving the rest of the scene in its original form. Furthermore, the third and final prototype maintains the functionality of the second prototype, but now each class of detected objects is assigned one of the two chosen styles for the work, specifically the works "Self-portrait" by Pablo Picasso and "O Gato" by Romero Britto. Finally, the performance of the developed solutions was considered in certain contexts to verify their effectiveness and adaptability. Notably, the solution demonstrated the ability to run at over 60 frames per second in the first prototype and around 10 frames in the third prototype, using a machine equipped with an RTX 2060 graphics card to execute the solution in both aforementioned cases.

Keywords: Computer vision. Python. Object detection. Neural style transfer. Machine learning. Convolutional neural networks.

SUMÁRIO

1 INTRODUÇÃO

Compreender o funcionamento e a elaboração de um aplicativo que utiliza Realidade Aumentada é crucial para entendermos a capacidade de tecnologias digitais no processamento, análise e modificação de imagens e vídeos digitais. Realidade Aumentada (RA) se refere a uma percepção direta ou indireta, em tempo real, de um ambiente físico do mundo real aprimorado/melhorado pela adição de informações virtuais geradas computacionalmente. Além disso, a Realidade Aumentada é, ao mesmo tempo, interativa e tridimensional enquanto combina objetos reais e virtuais (CARMIGNIANI; FURHT, 2011).

É notório o crescimento e importância da Realidade Aumentada para a indústria, como, por exemplo, o mercado de Realidade Aumentada e Realidade Virtual na área de saúde que em 2020 equivalia a dois bilhões de dólares e espera-se que possua uma taxa de crescimento anual composta de 27,2% entre os anos de 2021 e 2028 (Grand View Research, 2019). A importância da Realidade Aumentada não ocorre somente em áreas como a da Saúde, grandes empresas como a Apple também investem em RA para o entretenimento, demonstrando isso com o lançamento do seu dispositivo Apple Vision Pro, dispositivo focado em tarefas digitais com o intermédio da Realidade Aumentada (APPLE, 2023a).

A Visão Computacional desempenha um papel fundamental na Realidade Aumentada, por ter o objetivo de capacitar o computador a compreender o conteúdo de imagens digitais, como fotografias e vídeos, possibilitando assim a sobreposição de informações virtuais sobre essas representações visuais. No entanto, a Visão Computacional é mais do que isso, ela é um campo de estudo em constante evolução, buscando desenvolver técnicas que permitam aos computadores compreenderem plenamente o conteúdo das imagens digitais, como fotografias e vídeos (BROWNLEE, 2019). Exemplos notáveis de aplicações da Visão Computacional incluem classificar objetos em uma cena, o reconhecimento facial e até mesmo a detecção de objetos, um elemento central para o desenvolvimento desse trabalho. Ademais, esse campo de estudo também possui grande interesse por parte da indústria, pois conforme o Grand View Research, em 2021 o valor global do mercado de Visão Computacional equivalia a 11,22 bilhões de dólares com taxa de crescimento anual composta de 7% de 2022 até 2030(Grand View Research, 2021).

Além da visão computacional, outro ponto importante para a realidade aumentada é a aprendizagem de máquina, definida como um conjunto de métodos que podem automaticamente detectar padrões em dados e, com os padrões descobertos, prever dados futuros ou realizar outra atividade, que pode ser de tomada de decisão ou até de formas de coletas de dados, aumentando

assim a eficiência da análise dos dados (MURPHY, 2013). Contudo, Utilizaremos de visão computacional e aprendizagem de máquina em conjunto, para a elaboração de uma sequência de experimentos visando uma solução com realidade aumentada.

Portanto, o **objetivo geral** do trabalho é propor uma arquitetura que englobe Realidade Aumentada, detecção de objeto e transferência neural de estilo por intermédio de algoritmos e técnicas avançadas de aprendizado de máquina. É importante salientar que essas tecnologias serão utilizadas e integradas no decorrer do desenvolvimento de protótipos que utilizarão pelo menos uma ou todas as tecnologias citadas anteriormente.

Para isso, é fundamental estabelecer **objetivos específicos** bem definidos. Nesse contexto, é imprescindível a realização de uma revisão bibliográfica detalhada sobre as ferramentas e tecnologias disponíveis pertinentes para a elaboração do trabalho proposto. Com o levantamento da revisão bibliográfica, é importante utilizar um conjunto de algoritmos de aprendizado de máquina que permita a detecção de objetos em tempo real e a possível utilização de técnicas de aprendizagem de máquina, como a transferência neural de estilo, em diversas cenas. Com a criação adequada dos algoritmos, ocorre a integração deles para que a solução aborde devidamente tais funcionalidades propostas. Além disso, é necessário avaliar a solução em sua aplicação funcional, garantindo que ela atenda plenamente as necessidades do usuário e ofereça uma experiência satisfatória.

Os protótipos propostos possuem várias áreas de aplicações, como, por exemplo, na manipulação de imagens digitais, que podem ser em tempo real ou não, por designers e editores de vídeo. Além de que, ao manipular o código, desenvolvedores de softwares podem entender melhor o funcionamento de tais tecnologias e como fazê-las funcionarem em conjunto. Contudo, a utilização dos protótipos não se restringe exclusivamente a especialistas técnicos e profissionais, podendo ser utilizado pelo público geral, assim como ocorre com filtros animados nas redes sociais.

Existem trabalhos relacionados que abordam tópicos semelhantes ao presente estudo. Por exemplo, em um trabalho anterior ((KADISH *et al.*, 2021)), foi explorada a utilização da transferência neural de estilo para criar um conjunto de dados capaz de ensinar um algoritmo a identificar pessoas em obras de arte. Nesse estudo, foi desenvolvido um código que gerou um conjunto de dados por meio da transferência neural de estilo, usando como base outro conjunto de dados conhecido como COCO. Vale ressaltar que o COCO é composto por imagens fotográficas, o que significa que não inclui imagens pintadas, desenhadas ou geradas artificialmente.

Consequentemente, quando um modelo treinado com o COCO se depara com imagens de obras de arte, pode enfrentar dificuldades na identificação de objetos nelas.

Além de criar o conjunto de dados, o código permitiu também treinar e testar um modelo de detecção. O repositório do código pode ser acessado através deste link: https://github.com/dkadish/Style-Transfer-for-Object-Detection-in-Art. Essa abordagem oferece a capacidade de detectar objetos em imagens que não são fotográficas, como as encontradas em museus e em bancos de dados online. Embora o enfoque principal do trabalho tenha sido a detecção de pessoas, é importante ressaltar que a mesma metodologia pode ser aplicada à detecção de outras classes de objetos (KADISH *et al.*, 2021).

Outro trabalho que, assim como o presente trabalho, mesclou o uso de detecção de objeto com realidade aumentada foi o trabalho de (LE *et al.*, 2021), nesse trabalho os autores criaram um conjunto de dados para detecção de cards por meio de uma aplicação em um dispositivo iOS, para sobrepor via câmera do dispositivo, recursos digitais em realidade aumentada na exata localização desses cards detectados.

2 REFERENCIAL TEÓRICO

O objetivo do capítulo se estabelece em apresentar a fundamentação teórica que embasa o trabalho a ser desenvolvido, além de fornecer um contexto teórico para análise e discussão dos resultados. Logo apresentaremos os conceitos de Visão Computacional, Realidade Aumentada, *Object Recognition* e Transferência Neural de Estilo. Estes conceitos serão importantes na execução das experimentações propostas.

2.1 Visão Computacional

Visão computacional é um campo de estudo que visa compreender os conteúdos de imagens digitais extraindo informações delas, podendo ser um objeto específico, uma descrição textual ou um modelo tridimensional, por meio de métodos que tentam reproduzir a capacidade da visão humana. Além disso, visão computacional é um campo multidisciplinar que pode ser referenciado como um subcampo da inteligência artificial e da aprendizagem de máquina (BROWNLEE, 2019).

O desafio da área consiste, de forma bastante simplificada e objetiva, em auxiliar os computadores a terem as mesmas percepções que os seres humanos têm ao visualizar uma imagem fotográfica ou vídeo. Pode aparentar ser fácil fazer o computador adquirir informações a partir de uma imagem, pois isso é uma característica inerente a quase todos os seres humanos, mas a visão humana é extremamente complexa. E mesmo após décadas de estudos, a visão computacional ainda se encontrava sem solução, pelo menos sem soluções para alcançar algo parecido com as capacidades visuais humanas (BROWNLEE, 2019). Esse desafio é aumentado pela nossa incapacidade de entender como a nossa visão funciona, mesmo com estudos de biologia com foco em entender seu funcionamento e a interpretação do cérebro para a informação recebida pela visão, mas como qualquer estudo que envolve cérebro, há muitos estudos a serem feitos sobre (BROWNLEE, 2019).

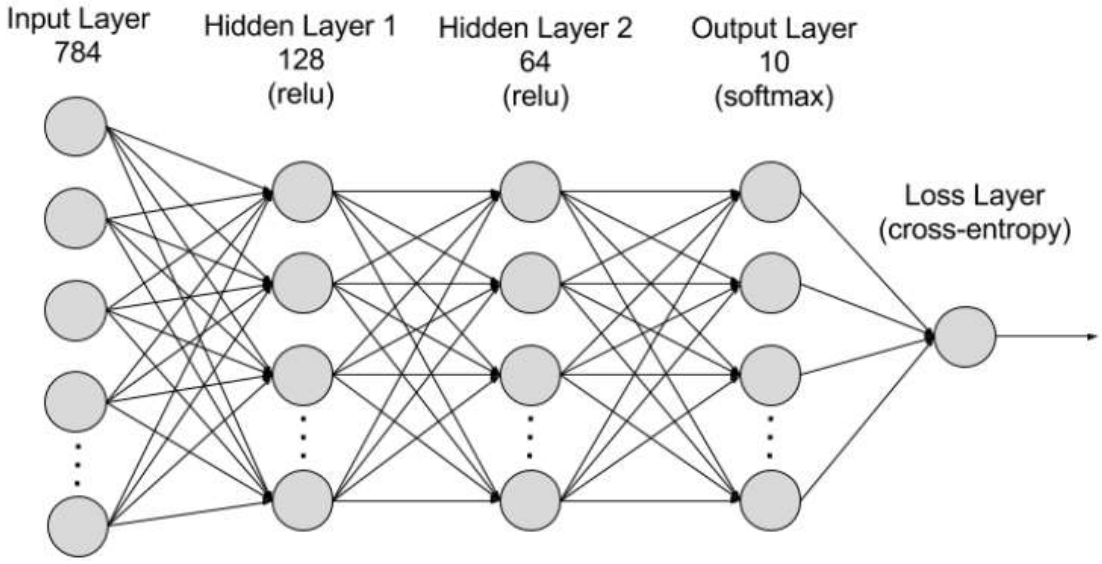
Todavia, houve muito progresso no campo, especialmente aplicado a sistemas de segurança automotiva, biometria para reconhecimento de digitais e até mesmo reconhecimento facial presente em *smartphones* e câmeras (BROWNLEE, 2019). As mais populares soluções que utilizam visão computacional envolvem reconhecimento de objetos em fotografias ou vídeos, como, por exemplo, classificação, identificação ou verificação de objetos em uma mídia visual digital (BROWNLEE, 2019).

2.2 Redes Neurais Artificiais

Existem diversas formas de se implementar elementos de Visão Computacional, uma forma bastante comum é a aplicação das Redes Neurais Artificiais (RNA). As Redes Neurais Artificiais são um tipo de processo de aprendizagem de máquina que utiliza nós ou neurônios interconectados em camadas, baseados numa simplificação matemática do funcionamento do cérebro. Também são sistemas adaptativos que permitem aos computadores aprender com erros e melhorar-se constantemente. Dentre suas capacidades, podemos citar o fato das Redes Neurais poderem resumir documentos e reconhecer rostos (AMAZON, 2023). Para fins de exemplificação, outra aplicação que utiliza de Rede Neural, assim como a Visão Computacional, seria o processamento de linguagem natural (PLN) o qual é a capacidade de processar textos criados por seres humanos assim como faz o ChatGPT [1] e aplicações semelhantes, contudo não iremos nos aprofundar em PLN, pois não é o foco do trabalho em questão.

O funcionamento de uma Rede Neural básica terá neurônios artificias interconectado em três camadas, como demonstra a Figura 1.

Figura 1 – Camadas de uma Rede Neural Artificial básica.



Fonte: https://aws.amazon.com/pt/what-is/neural-network/

A camada de entrada é responsável por receber as informações do mundo externo. Nessa camada se situam os nós de entradas que processarão os dados, analisarão ou categorizarão esses dados e os encaminham para a próxima camada por meio dos nós. As próximas camadas

[1] ChatGPT é um modelo de linguagem desenvolvido pela OpenAI que pode gerar texto coerente e contextual. Disponível em https://openai.com/chatgpt. Acesso em: 20 de nov. de 2023

são conhecidas como camadas ocultas, essas camadas usam as entradas geradas pela camada de entrada ou de outras camadas ocultas. Cada camada oculta analisa o resultado da camada anterior, processa-o ainda mais e o encaminha para a próxima camada. Por fim, essas entradas são processadas até chegar na camada de saída, essa camada é responsável por fornecer o resultado de todos os dados processados pela rede neural artificial.

A última camada pode ter um ou vários nós, dependendo do tipo de resultado que ela será responsável por fornecer no final do processamento. Ou seja, em um cenário no qual a Rede Neural tem como resultado a existência ou não de um objeto em cena, só seria necessário a existência de um só nó na camada de saída, pois o resultado seria único: falso ou verdadeiro (0 ou 1). Mas caso a Rede Neural seja responsável por fornecer a classe do objeto presente em uma cena, seria necessário mais nós, pois cada nó demonstraria o valor relativo à possível classe do objeto na cena recebida na camada de entrada.

As Redes Neurais também podem ser profundas, isso ocorre quando a Rede Neural possuir várias camadas ocultas, com milhões de neurônios/nós interligados. Nesse tipo de Rede Neural, há métricas chamados pesos que representam as conexões entre um nó e outro. Caso o peso entre os nós sejam positivos, eles terão afinidades, contudo, se tiverem peso negativo se reprimirão. Nós com maiores pesos tem mais influência nos outros nós. Dado que Redes Neurais Profundas possuem milhões de nós na sua estrutura, para seu devido funcionamento, ela exige de muito mais treinamento, com milhões de exemplos de dados de treinamento, enquanto outras Redes Neurais simples precisam de apenas centenas ou milhares (AMAZON, 2023).

Os tipos das Redes Neurais são categorizados com base em como os dados fluem do nó de entrada para o nó de saída. dentre os tipos se pode citar as Redes Neurais Feedforward, no qual o processamento dos dados só ocorrem em uma direção, do nó de entrada para o nó de saída. Nesse tipo de Rede Neural, cada nó de uma camada está conectado com todos os nós da próxima camada.

Outro tipo é o tipo com Algoritmo de Backpropagation. Esse tipo de Rede Neural aprimora o resultado aumentando o peso entre os nós que possuem valores mais precisos e diminuindo os pesos entre os nós que resultaram em valores não esperados na camada de saída. Isso é possível pois esse algoritmo age da camada de saída até a camada de entrada.

Por fim, outro tipo de Rede Neural, que será de suma importância para esse trabalho, por serem muito úteis para a classificação de imagens, é a Rede Neural Convolucional. Nesse tipo de Rede Neural, as camadas ocultas executam funções matemáticas específicas, como resumo ou

filtragem. Dessa forma, é possível extrair recursos relevantes das imagens. Cada camada oculta extrai e processa recursos diferentes como bordas, cores e profundidade (AMAZON, 2023).

2.3 Realidade Aumentada

Umas das definições mais amplas e aceitas é a proposta por Azuma em 1997, de acordo com ele, a realidade aumentada deve seguir as seguintes três características (AZUMA, 2001):

1. Combinar objetos reais e virtuais em um ambiente real;
2. Funcionar interativamente em tempo de execução;
3. E alinhar objetos virtuais e reais entre si;

Com realidade aumentada, as informações digitais farão parte da realidade, complementando-o, pelo menos na percepção do usuário. Enquanto na realidade virtual o usuário estará completamente imerso em um ambiente gerado computacionalmente, sem qualquer interação com informações do mundo real (SCHMALSTIEG; HöLLERER, 2016).

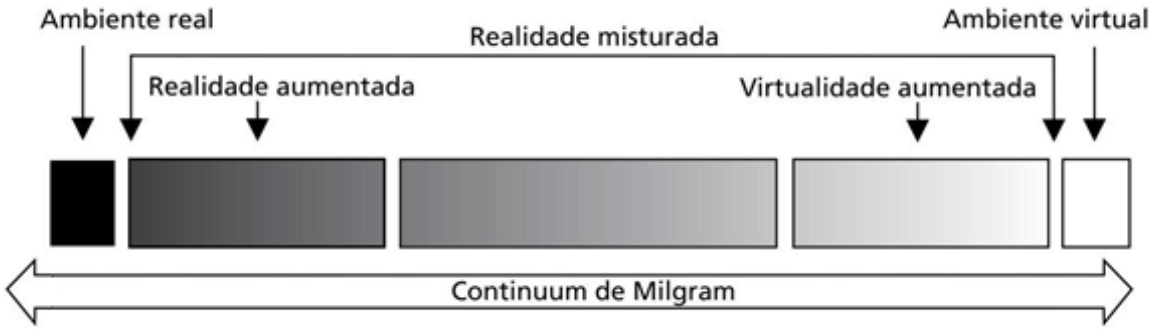
A realidade aumentada está se tornando cada vez mais popular para aprimorar jogos computacionais. Dispositivos como o Oculus Rift da Meta e o HoloLens da Microsoft recebem uma grande atenção do público e, por conseguinte, do mercado focado em realidade Aumentada (SCHMALSTIEG; HöLLERER, 2016). A ingressão mais a fundo em RA de grandes empresas como a Apple, comprovam esse ponto (APPLE, 2023a).

A realidade aumentada encontra-se mais próxima do ambiente real, ou seja, é adicionado um objeto virtual em torno de um ambiente real, isso é exemplificado no conceito de Continuum Realidade-Virtualidade de Milgram, o qual é um espectro que situa as tecnologias que perpassam entre o ambiente real e o ambiente virtual. Do Continuum, podemos citar outros objetos de estudos como a virtualidade aumentada, que se encontra mais próximo da realidade virtual, ou a realidade misturada, que engloba as duas formas de visualização de dados do mundo real e virtual por meio da realidade aumentada e da virtualidade aumentada, como mostrado na Figura 2.

Apesar de que, em um primeiro momento, ser comumente alinhado com a experiência de Realidade Aumentada para algo predominantemente visual, (AZUMA, 2001) abrange a sua definição não somente para aquilo que conseguimos ver, mas também para o que sentimos com os outros sentidos.

Há diversas formas para aplicação de uma realidade aumentada, alguns pesquisadores

Figura 2 – Continuum Realidade-Virtualidade de Milgram.



Fonte: https://www.researchgate.net/figure/Continuo-Real-Virtual-adaptado-de-Milgram-and-Kishino-1994-e-Tori-and-da-Silva_fig1_342298366

se referem a remover um objeto de um ambiente real para sobreposição de um objeto virtual como realidade mediada ou realidade diminuída, mas para Azuma(AZUMA, 2001), essa característica de remoção de objeto seria vertente da Realidade Aumentada e não uma definição totalmente distinta.

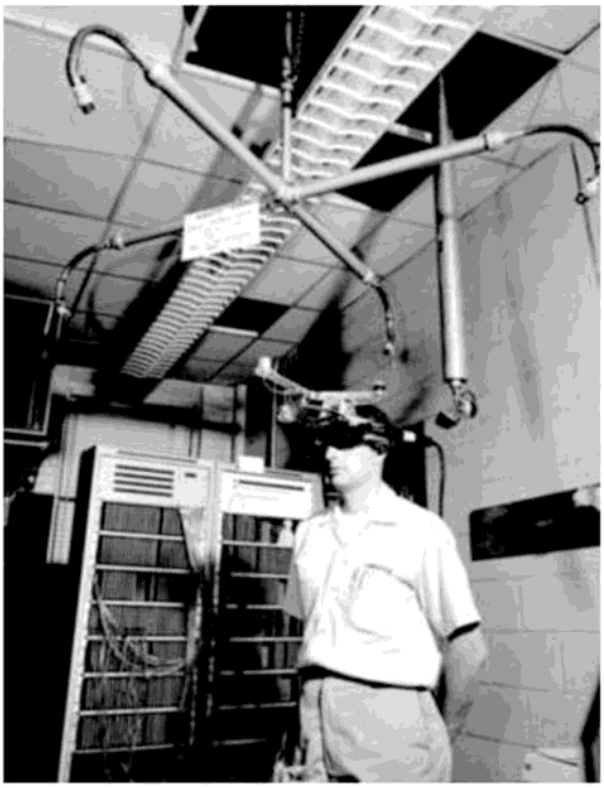
2.3.1 História

O começo da Realidade Aumentada, como é conhecida atualmente, ocorreu na década de 60 em um trabalho de Ivan Sutherland (ver figura 3), o qual usou um *head-mounted display* ou *head-worn display* (HMD) um dispositivo tecnológico semelhante a um óculos que possui uma tela capaz de providenciar imagem em frente aos olhos do usuário (AZUMA, 2001).

Apesar do seu início na década de 60, apenas na década de 90 que o interesse pela tecnologia cresce, somando pesquisas o suficiente para que RA pudesse ser caracterizada como um campo de pesquisa. Na mesma década, mais precisamente em 1997, Azuma publicou uma pesquisa que define o campo, além de descrever diversos problemas e resumir a evolução da tecnologia. A partir disso, o crescimento da Realidade Aumentada é notável, constando com vários eventos, incluindo Workshop e Simpósios internacionais para RA e também para Realidade Mista. Além de eventos, surgem organizações focadas na tecnologia, onde se destaca Mixed Reality Systems Lab no Japão e Arvika Consortium in Germany. Todos esses eventos aconteceram ainda na década de 90, assim como o surgimento dessas companhias (AZUMA, 2001), demonstrando o interesse em comum do mundo na época e na atualidade (Grand View Research, 2021).

Contudo, com o crescimento da velocidade disponível para redes sem fios e a rápida evolução tecnológica, pesquisadores tiveram avanços no design de HMD, possibilitando

Figura 3 – Sword of Damocles, por Ivan Sutherland (Primeiro HMD com Realidade Virtual).



Fonte: https://www.researchgate.net/figure/Sword-of-Damocles-by-Ivan-Sutherland-First-computer-connected-VR-system_fig3_348233163

melhorias em tecnologia óptica, microdisplays e miniaturização de eletrônicos. Porém, o Helmet Display (HMD) não é a única categoria de dispositivos de visualização utilizada para a realidade aumentada, pois além dela existem mais duas outras categorias de dispositivos de visualização principais para a atividade de Realidade Aumentada(CHEN *et al.*, 2019), as quais são:

1. A categoria de displays portáteis ou *handheld device display*, categoria de dispositivo que são muito leves, pequenos e sua popularidade se deve principalmente por conta dos *smartphones*, por meio da perspectiva de filmagem de vídeo de tais dispositivos.
2. A categoria de outros dispositivos com telas, como, por exemplo, telas de computadores pessoais, que podem manipular a cena do mundo real combinando o que é capturado por uma câmera com um modelo tridimensional gerado pelo computador e transmitido para um monitor de mesa.
3. A categoria de tecnologia de registro 3D, esse tipo de tecnologia permite que imagens virtuais sejam projetadas/sobrepostas com precisão no ambiente real.
4. A categoria de interação inteligente, nessa categoria ocorre a relação com os tipos citados anteriormente. Contudo, além desse relacionamento, ocorre a interação por dispositivos hardwares, por localização ou outra forma. Com o desenvolvimento dessa categoria,

a realidade aumentada não apenas sobrepõe informações virtuais a cenas reais, como também permite a interação entre pessoas e objetos virtuais em cenas reais. Ou seja, caso ocorra uma interação com algum objeto virtual, esse objeto dará algum feedback, permitindo que o público da aplicação tenha uma melhor experiência.

2.3.2 Frameworks e bibliotecas

Atualmente, existem muitas ferramentas que automatizam tarefas de RA e implementam diversos algoritmos que podem ser utilizados para se construir aplicações utilizando essas tecnologias. Nessa subseção, serão listadas ferramentas utilizadas na elaboração de soluções em realidade aumentada e de forma cronológica.

2.3.2.1 ARToolKit

ARToolKit é uma biblioteca de Realidade Aumentada desenvolvida em 1999 por Hirokazu Kato no Instituto de Ciência e Tecnologia Nara e lançado pela Universidade de Washington HIT Lab. Basicamente é uma biblioteca de computação visual para rastreamento que permite o usuário criar aplicações em realidade aumentada. Seu funcionamento se baseia na capacidade de calcular em tempo real a posição real da câmera e a sua orientação relativa a marcadores físicos para que, com essas informações, rastrear e posicionar objetos virtuais em um espaço tridimensional. Portanto, a partir do momento em que a posição real da câmera é calculada, uma câmera virtual pode ser posicionada na mesma exata posição e modelos gráficos computacionais tridimensionais podem ser desenhados para sobrepor os marcadores (FURHT B.; LIVINGSTON, 2011).

Com o sucesso do ARToolKit, foi criada uma versão estendida do mesmo chamada de ARToolKitPlus, no entanto, em 2011 já não estava mais sendo desenvolvida e possuía um sucessor: Studierstube Tracker. (CARMIGNIANI; FURHT, 2011)

2.3.2.2 Studierstube Tracker

O conceito do Studierstube Tracker é muito parecido com o do seu antecessor ARToolKitPlus, que por sua vez era também muito parecido com a sua versão mais básica, o ARToolKit. Contudo, ao contrário do ARToolKit e ARToolKitPlus, o Studierstube Tracker possuía um código base completamente diferente e não era open-source, fazendo-o não ter seu

código disponível para livre uso.

Ele suporta telefone móveis e na sua versão Embedded System (ES), também suporta computadores pessoais, característica essa que trouxe muitas vantagens ao Studierstube Tracker como necessitar somente de 100 KB (algo em torno de 5 a 10 porcento do que era necessário para o ARToolkitPlus) e capaz de um processamento extremamente rápido, conseguindo alcançar o dobro da capacidade de processamento do ARToolKitPlus em dispositivos móveis e capaz de chegar a 1ms por frame nos computadores (CARMIGNIANI; FURHT, 2011).

2.3.2.3 ARKit

Studierstube Tracker e Studierstube ES suporta a plataforma do Iphone, no entanto, não são open-sources, o que é um empecilho para os desenvolvedores da plataforma. Para auxiliar os desenvolvedores do Iphone, a Apple cria o framework ARKit, o qual é um conjunto de classes que permitem aos desenvolvedores implementarem realidade aumentada em qualquer aplicação do Iphone sobrepondo informações, que geralmente são geográficas na visão da câmera (CARMIGNIANI; FURHT, 2011).

Atualmente o ARKit está na sua sexta versão, permitindo vídeos em definição 4K, suporte a vídeo HDR e captura de imagens de fundos em alta resolução. Entre as suas características podemos citar a geometria da cena que permite criar um mapa topológico do espaço capaz de identificar o piso, parede, telhado, janelas, portas, etc. (APPLE, 2023b).

Os dispositivos como o Iphone 12 Pro e 12 Pro Max ou até mesmo o iPad Pro de 11 polegadas, possuem a tecnologia LiDAR integrada, essa tecnologia é um acrônimo para *Light Detection and Ranging* e seu funcionamento basicamente se resume a emitir um laser por um transmissor e essa luz será refletida por objetos em cena. A luz refletida será detectada pelo sistema que saberá o tempo de voo ou *time of flight*(TOF) e com isso será descoberta a distância para o objeto em cena atingido pelo laser (SYNOPSYS, 2023). Com essa tecnologia, o ARKit consegue detecções de planos extremamente rápidas, permitindo a colocação instantânea de objetos de Realidade Aumentada no mundo real sem a necessidade de scanner (APPLE, 2023b).

2.3.2.4 ARCore

ARCore é a plataforma do Google para criação de experiências em realidade aumentada. Ele permite várias funcionalidades para os *smartphones*, como detectar o ambiente e entender o mundo ao redor, além de permitir que o usuário interaja com as informações. Apesar

de ser do Google, o qual foi o comprador da empresa responsável pelo Android, e ter como principal rival de mercado do Android a Apple, o ARCore é disponível para dispositivos da Apple, além de também ser disponível para o Android (GOOGLE, 2023c).

Seu funcionamento acontece essencialmente em acompanhar a posição do seu dispositivo móvel à medida que ele se move e a partir disso ter o seu próprio entendimento de mundo, para isso ele utiliza de três recursos principais (GOOGLE, 2023c):

1. O acompanhamento de movimento, que permite que o smartphone entenda e monitore a posição dele em relação ao mundo real. Para isso ele utiliza de recursos, algo parecido de certa foma com os marcadores físicos do ARToolKit. Com os recursos rastreados, o ARCore verifica como esses pontos se movem ao longo do tempo. Contudo, combinando o movimento dos pontos e as leituras inerciais do smartphone, o ARCore determina a posição e a orientação do smartphone conforme ele se move no espaço.
2. A compreensão ambiental, que permite que o smartphone detecte e entenda o tamanho e local de todos os tipos de superfícies.
3. A estimativa de luz que permite que o smartphone estime as condições de iluminação atuais do ambiente.

O ARCore encontra-se atualmente na versão 1.37.0 e oferece recursos como semântica de cena que é um recurso que facilita a compreensão do mundo ao redor do usuário, por oferecer detalhes mais minuciosos sobre a cena ao redor. Outro recurso interessante são as âncoras na cobertura, os quais são um novo tipo de âncora geoespacial que ajuda a ancorar conteúdo em uma cobertura (GOOGLE, 2023b). Âncoras geoespacial é a localização do item no mundo real a ser utilizado pela tecnologia, seja para sobrescrever ou colocar numa posição relativa ao item do mundo real com um item virtual.

2.3.3 Comparação entre os frameworks de realidade aumentada

Dentre os frameworks para realidade aumentada citados, somente o ARToolKit, ARKit e ARCore estão atualmente recebendo atualizações. Apesar do ARToolKit permitir o desenvolvimento de software para uma vasta gama de plataformas como IOS, Android, macOS, Windows e Linux (VAUGHAN; LAMB, 2023), o ARCore e o ARKit possuem o respaldo das maiores gigantes tecnológicas existentes no momento, Google e Apple, respectivamente. Esse fato contribui para maior qualidade de documentação para o desenvolvimento e uma comunidade maior e mais ativa.

No entanto, o ARKit, como muitos outros produtos da Apple, restringe o acesso aos seus softwares para dispositivos Apple, o que pode ser limitante se não houver dispositivos Apple disponíveis para o desenvolvimento. Portanto, no trabalho em questão, optou-se pelo ARCore como framework escolhido. A escolha se baseou na capacidade de testar o aplicativo em dispositivos Android e, ao mesmo tempo, no fato de o ARCore também oferecer funcionalidades que funcionam adequadamente em dispositivos iOS.

2.3.4 Aplicações com Realidade Aumentada

Nesta seção serão listadas aplicações que utilizaram da tecnologia de realidade aumentada, sintetizando o funcionamento básico de cada uma delas.

Tiveram vários jogos que quiseram utilizar da realidade aumentada para incrementar a imersão dos usuários para com o jogo. Entre os mais famosos podemos citar o ARQuake lançado em 2002, versão de realidade aumentada do famoso jogo de tiro em primeira pessoa Quake. Para o jogo, cada jogador deveria vestir um sistema de realidade aumentada que consistia em um notebook posicionado em uma mochila que possuía vários sensores (como demonstrado na Figura 4) para rastrear a posição, incluindo um *Global Positioning System* (GPS) para orientação e uma câmera também para o rastreamento de visão computacional baseado em marcadores (GU; DUH, 2011).

Dois anos depois foi lançado o AR-Soccer, o primeiro jogo em realidade aumentada portátil (GU; DUH, 2011). AR-Soccer é um jogo de smartphone que o jogador podia chutar uma bola virtual em uma trave virtual e, com a utilização da câmera do dispositivo móvel, o chute ser calculado pelo rastreamento realizado pela câmera do dispositivo (ver figura 5).

Seu funcionamento era visto pelos seus criadores como uma variante do famoso jogo Pong. Contudo, por limitação dos dispositivos da época, o jogo enfrentava problemas por conta da baixa quantidade de quadros durante seu funcionamento (cerca de 5 a 7 quadros por segundo) exigindo que os jogadores chutassem lentamente para que seu movimento fosse calculado pelo algoritmo (PAELKE *et al.*, 2004).

A lista de jogos é grande, porém poucos podem se comparar com o sucesso estrondoso que teve o Pokémon GO. Nas primeiras semanas de julho de 2016, o jogo começou a ser baixado em vários países e começou a ser um sucesso mundial. Seu sucesso não se resume a apenas a uma boa implementação por parte dos desenvolvedores ou pela simplicidade de seu uso, a franquia Pokémon se expandia pela cultura de diversos países e se estabelecia como um

Figura 4 – Jogadores utilizando os itens necessários para jogar ARQuake

Fonte: https://www.tinmith.net/arquake/

caso de sucesso por décadas (HJORTH; RICHARDSON, 2017).

O jogo, assim como a sua série animada, se passa em um mundo no qual existe monstrinhos que os humanos os procuram, os capturam e os utilizam em batalhas. Tendo isso mente, a funcionalidade se resumia em andar no mundo real, passar próximo a um desses monstrinhos e capturá-los para colecioná-los e treiná-los para serem utilizados em batalhas. A realidade aumentada é utilizada na parte de captura desses bichinhos, pois o aplicativo utiliza a câmera para mostrar o monstro a ser capturado sobreposto ao mundo real capturado pela câmera do smartphone (G1, 2015).

2.4 Object Recognition

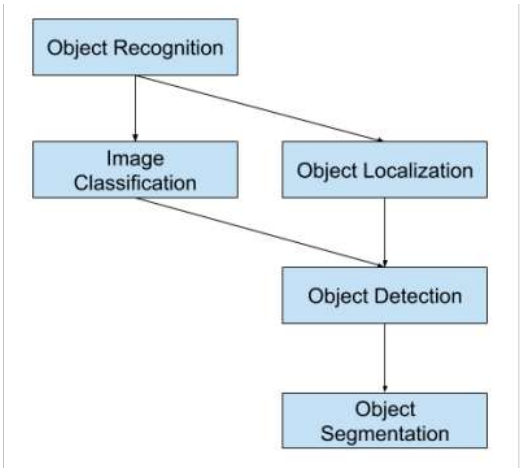
Object Recognition é um termo geral que abrange uma coleção de tarefas relacionadas a visão computacional que envolve, como o nome sugere, o reconhecimento de objetos digitais em uma mídia visual digital (BROWNLEE, 2019) (ver figura 6).

Dentre as tarefas englobadas pelo *Object Recognition*, temos as tarefas de:

Figura 5 – A trave do gol e a bola virtual no jogo AR-Soccer

Fonte: (GU; DUH, 2011)

Figura 6 – Visão geral das tarefas de Visão Computacional de Object Recognition



Fonte: (BROWNLEE, 2019)

1. *Image Classification*, cujo objetivo é prever o tipo ou a classe de um objeto na imagem, esperando como entrada para processamento uma imagem com objetos para serem classificados e como saída do processamento uma etiqueta indicando a classe do objeto reconhecido.
2. *Object Localization*, que localiza a presença do objeto em uma imagem e indica a sua

localização por meio de uma caixa ao redor da imagem localizada. Ou seja, essa tarefa tem como entrada uma imagem com objetos e terá como resultado caixas que delimitam o objeto localizado.

3. *Object Detection*, seria a junção do *Image Classification* e do *Object Localization*, portanto essa tarefa tem o objetivo de localizar a presença de um objeto em uma imagem digital e delimitar sua posição.
4. Além dessas tarefas, existe o *Object Segmentation*, também chamado de *Object Instance Segmentation* ou *Semantic Segmentation*. Nesse tipo de tarefa os objetos são destacados na exata silhueta do objeto, por meio de seus pixels, ao invés de uma caixa em torno do objeto, como acontece com o *Image Classification*.

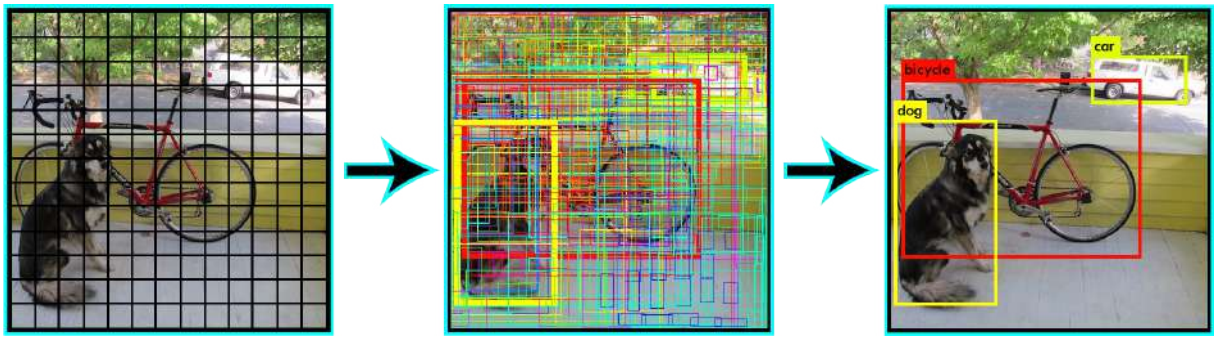
2.4.1 YOLO: You Only Look Once. Detecção de objeto em tempo real

Nessa subseção será tratado a história e desdobramentos acerca da ferramenta de detecção de objeto YOLO. Essa ferramenta surgiu com números tão bons quanto outras ferramentas já estabelecidas como SSD, Faster R-CNN, RetinaNet, etc. Apesar da gama de ferramentas no mercado, a escolha do YOLO para o trabalho em questão se deve por ser uma das mais velozes do mercado de detecção de objeto, além de manter alta taxa de precisão, dando margem de desempenho para aplicar novas funcionalidades à detecção de objeto por meio do desenvolvimento desse trabalho. Além disso, o YOLO é bem documentado, o que é de grande ajuda para o desenvolvimento que utilizará essa ferramenta.

Em 2015, Joseph Redmon, Santosh Divvala, Ross Girshick e Ali Farhadi apresentam ao mundo YOLO uma nova abordagem para detecção de objeto, o *You Only Look Once* (YOLO). O YOLO se destaca dentre as outras ferramentas de detecção de objeto, pois, ao contrário dos métodos de detecção de objeto já existentes que geralmente necessitam localizar cada um dos objetos em cenas e identificá-los posteriormente, o YOLO dividia a imagem que seria utilizada para detecção de objeto em pequenas caixas inferindo a probabilidade das possíveis classes dos objetos em cenas em cada uma das caixas dividas anteriormente, o que é bem mais rápido em termos de desempenho (ver figura 7) (REDMON *et al.*, 2016).

A figura 7 demonstra o funcionamento do YOLO, ele divide a cena que está sendo processada em pequenos quadrados, cada quadrado terá uma classe que será definida pelo YOLO. No final do processamento, a rede neural juntará os quadrados de mesma classe e definirá que até os limites desses quadrados está todo o objeto da classe detectada, que no caso foi o cachorro, a

Figura 7 – Funcionamento do YOLO para detecção de objeto



Fonte: (REDMON; FARHADI, 2016)

bicicleta e o carro, demonstrados pela indicação quadrangular de cor amarela, vermelha e verde, respectivamente.

Além dessa forma de detecção do YOLO, outro ponto que favorece a otimização diretamente no desempenho de detecção é o fato de utilizar somente uma rede neural profunda chamada de Darknet(REDMON; FARHADI, 2016). Rede neural profunda, em suma, é uma rede neural com várias camadas ocultas (camadas que ficam entre a camada de entrada e a camada de saída) com milhões de neurônios artificiais (AMAZON, 2023). Contudo, Darknet é um framework de rede neural de código aberto escrito em C e CUDA que o grupo responsável pelo framework também é conhecido pelo mesmo nome.

A fins de comparação, outro modelo de detecção de objeto como o R-CNN necessitava de milhares de redes neurais para uma única imagem, permitindo o modelo YOLO ser mais de mil vezes mais rápido que o R-CNN (REDMON; FARHADI, 2016), um ponto interessante sobre os dois modelos é que o Ross Girshick, desenvolvedor do R-CNN, foi também um dos autores e contribuidores do YOLO e depois se tornou pesquisador de Inteligência Artificial para o Facebook (BROWNLEE, 2019). No lançamento do YOLO, o seu modelo base tinha capacidade de processar imagens em tempo real a 45 quadros por segundo e na sua versão menor chamada de Fast YOLO conseguia processar incríveis 155 quadros por segundo. O YOLO, comparado com sistemas de detecção de última geração, errava mais a localização de objetos, porém era menos suscetível a predizer falsos positivos em objetos ao fundo da imagem (REDMON *et al.*, 2016).

Joseph Redmon, o principal nome no desenvolvimento do YOLO ficou responsável pelas próximas versões até o YOLOv3, mas por questões morais e éticas optou em parar o desenvolvimento da ferramenta (YUAN, 2020). Após a saída de Redmon, o YOLO passou na mão de outras empresas como a chinesa Meituan, mas atualmente a Ultralytics que está

responsável por ele.

YOLOv8 é a última versão existente do YOLO e será utilizada no trabalho, é uma ferramenta estado da arte feito com base nos seus antecessores. Lançada em 12 de janeiro de 2023, essa versão teve uma série de melhorias de desempenho e funcionalidades, dentre as funcionalidades existentes, além das já citadas como a detecção de objeto, classificação de objeto e a segmentação de objeto, nessa versão podemos também rastrear o objeto que quisermos e também estimar a pose de um objeto em cena.

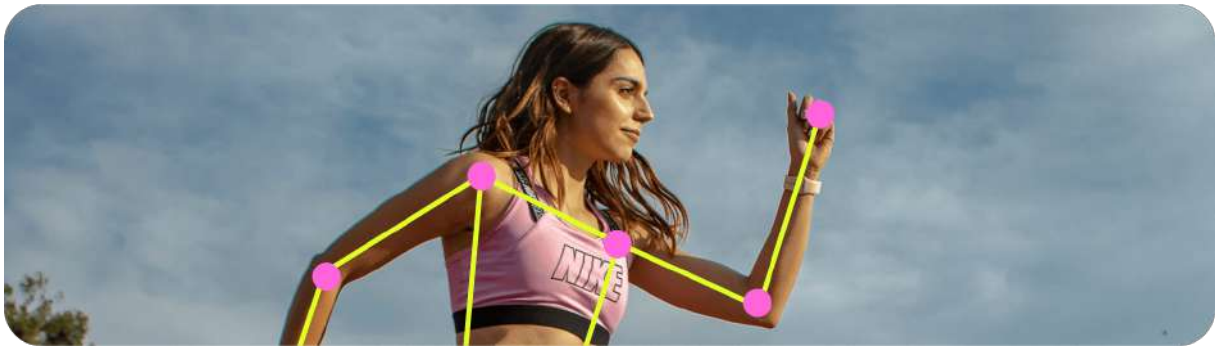
Na funcionalidade de rastreio ou *track*, o YOLO permite identificar a localização do objeto e classificá-lo associando-o a um identificador que será responsável por traçar um caminho onde o objeto identificado passou (ULTRALYTICS, 2022) (ver figura 8).

Figura 8 – Funcionamento da tarefa de rastreio do YOLOv8

Fonte: https://docs.ultralytics.com/modes/track/

Além de rastreio, outra funcionalidade presente no YOLOv8 é a de estimativa de pose, esse recurso envolve identificar a localização de pontos específicos de uma imagem, esses pontos são chamados de pontos-chave. Esses pontos-chave podem representar várias partes do objeto como juntas, pontos de referências, etc. As localizações desses pontos-chave são representados por coordenadas (ULTRALYTICS, 2023) (ver figura 9).

Figura 9 – Funcionamento da funcionalidade de estimativa de pose do YOLOv8



Fonte: https://docs.ultralytics.com/tasks/pose/

Contudo, dentre as funcionalidades presentes no YOLOv8, para o projeto destacamos o modelo de segmentação (ver figura 10), que consta no YOLO desde a sua quinta versão. *Object segmentation*, ou modelo de segmentação, como dito anteriormente, é um processo que, além de detectar o objeto, a silhueta do objeto detectado é destacada. Esse será a tarefa utilizada para o desenvolvimento de nossos protótipos.

Figura 10 – Funcionamento de um modelo de segmentação

Fonte: https://github.com/ultralytics/yolov5/releases

2.4.2 Transferência Neural de Estilo

A transferência neural de estilo é uma técnica que, normalmente, utiliza Redes Neurais Artificiais para aplicar estilos de arte visual de uma imagem para outras. Essa técnica permite transferir os estilos artísticos de uma imagem para outra, gerando resultados únicos e estilizados. Dessa forma, é possível combinar características visuais distintas e criar novas obras de arte digitais. O algoritmo dessa técnica foi originalmente proposto por um grupo de pesquisadores em um artigo intitulado “A Neural Algorithm of Artistic Style” publicado em 2015 (GATYS *et al.*, 2015).

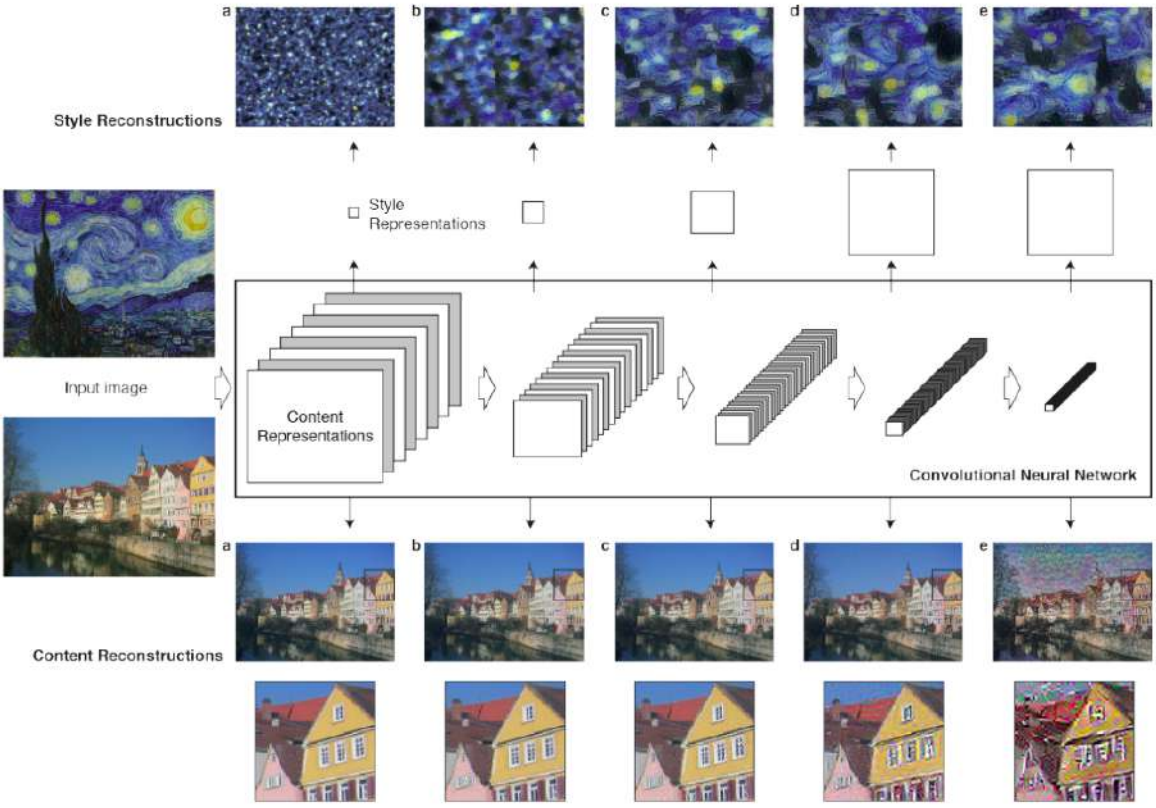
Redes Neurais Convolucionais ou *Convolutional Neural Network* (CNN) são a classe de Redes Neurais profundas mais poderosa na tarefa de processamento de imagens. CNN consiste em camadas de pequenas unidades computacionais que processam uma informação

visual de forma hierárquica por *feed-forward*, ou seja, as informações fluem somente em uma direção, e essa direção seria das camadas de entradas para as camadas de saída. Cada camada das unidades computacionais podem ser entendidas como uma coleção de filtros, no qual esses filtros extraem uma característica da imagem a ser processada. Além disso, cada camada tem como saída diferentes versões filtradas da imagem que está sendo processada (GATYS *et al.*, 2015).

Redes Neurais Convolucionais, quando treinadas para reconhecimento de objetos, elas desenvolvem a capacidade de aumentar a quantidade das informações sobre a imagem enquanto o processamento hierárquico de suas camadas ocorrem. Portanto, a cada camada que a imagem é processada, mais informações do conteúdo da imagem o algoritmo é capaz de captar, ao invés de somente captar os valores dos pixels. As camadas superiores das Redes capturam o conteúdo de alto nível em termos de objetos e sua disposição na imagem de entrada, enquanto que as camadas inferiores simplesmente reproduzem os pixels da imagem original (GATYS *et al.*, 2015) (ver figura 11).

Os pesquisadores utilizam do espaço de características, também conhecido como *feature space* das Redes Neurais Convolucionais, para obter a representação do estilo da imagem. O espaço de característica foi criado originalmente pelos mesmo pesquisadores em questão e para a captura de informações da textura da imagem a ser processada. Esse espaço de características é feito sobre o retorno de cada camada da Rede Neural (GATYS *et al.*, 2015). Espaço de características, para o aprendizado de máquina, são os conjuntos de representações numéricas extraídas de uma Rede Neural Convolucional, durante o processamento de uma imagem (GRIGG, 2019). Essas representações numéricas capturam diferentes aspectos da imagem como padrões e formas, mas no caso específico de (GATYS *et al.*, 2015), o espaço de característica capturava a textura da imagem e não os seus dados brutos.

Figura 11 – Funcionamento do algoritmo de Redes Neuras Convolucionais



Fonte: (GATYS *et al.*, 2015)

3 METODOLOGIA

O objetivo do projeto em questão é elaborar uma solução que englobe realidade aumentada, detecção de objeto e transferência neural de estilo e, além disso, seja interativa e imersiva para o usuário.

Para isso, a solução precisará de uma imagem de cunho artístico que será utilizada como imagem de entrada no algoritmo de transferência neural de estilo e passará as suas características artísticas para uma segunda entrada, além disso, a solução conseguirá processar dados visuais provenientes de dispositivos de filmagem integrados ao computador, vídeos ou imagens estáticas. A partir desses dados visuais, o sistema irá primeiramente identificar o objeto de interesse por meio do algoritmo do YOLOv8. Em seguida, será aplicada a transferência neural de estilo no objeto detectado pelo algoritmo do YOLOv8 para que as características artísticas da imagem escolhida seja aplicada somente no objeto detectado, mantendo o restante da imagem em sua forma original.

Para alcançar o objetivo, o desenvolvimento da solução seguirá as 7 etapas metodológicas demonstradas pela figura 12, que serão mais detalhadas a seguir.

Figura 12 – Passos metodológicos aplicados no projeto

Fonte: O autor

1. Pesquisa preliminar: foi realizada uma pesquisa aprofundada acerca das tecnologias de realidade aumentada, detecção de objetos e transferência neural de estilo. A pesquisa foi importante, por ajudar a embasar de forma teórica e prática o desenvolvimento da solução, por meio de levantamento de soluções que auxiliaram na elaboração da solução, além de códigos, fazendo com que fosse implementada funcionalidades necessárias para o projeto. Dentre as ferramentas obtidas pela pesquisa, destacou-se o YOLO, como ferramenta de detecção de objeto e de realidade aumentada. Também para Realidade Aumentada foi considerado o ARCore (GOOGLE, 2023b). Contudo, Para a atividade de Transferência Neural de Estilo foi utilizado o código de (CHOUDHARY, 2020) como base.
2. Arquitetura do sistema: nessa etapa, foi discutida a forma que a arquitetura seria, tendo em vista que ela tem que conseguir integrar componentes de realidade aumentada, detecção de objetos e transferência neural de estilo. Portanto, foi escolhida as ferramentas e bibliotecas apropriadas para a implementação considerandos pontos como desempenho e facilidade de uso. Portanto, foi necessário estudar mais a fundo as ferramentas levantadas no primeiro passo, focando no que elas eram capaz e no que podiam auxiliar no desenvolvimento da arquitetura dos protótipos.
3. Coleta e processamento de dados visuais: para essa finalidade foi utilizado o YOLO, tendo em vista que ele nativamente consta com uma ferramenta que permite várias fontes de coletas de dados visuais como, por exemplo, escolher usar um vídeo da internet, um vídeo local armazenado no dispositivo ou a webcam para a função de coleta de dados visuais.
4. Detecção de objeto: além de possuir formas nativas de coleta de dados visuais, o YOLO foi o responsável por fazer a detecção do objeto. Mais detalhadamente, foi utilizado a segmentação de objeto para o projeto, por ser necessário não somente a localização e nome do objeto detectado, mas também toda a silhueta dele, tendo em vista que somente os pixeis que abrange o objeto almejado terão que passar pela transferência neural de estilo nos próximos passos. É importante que a segmentação de objeto seja precisa e rápida, e o YOLO se propõe a ter essas características. Ademais, o YOLO dispõe de chamadas de retornos, ou *callbacks*, os quais são métodos que recebem como parâmetro informações da atividade que esta sendo executada. Esses métodos são acionados em certos pontos do ciclo de treinamento e detecção. Os pontos que acionam geralmente são no início ou no final de alguma tarefa da ferramenta, por exemplo, caso for necessário ter informações dos dados que estão sendo usados no pré-treinamento, bastava utilizar o método add_callback

contendo dois parâmetros, o primeiro com “on_pretrain_routine_start” e o segundo com um método anteriormente criado, esse método recebido no segundo parâmetro constará com um parâmetro e nesse parâmetro ele terá as informações da atividade que está sendo executada, como o objeto detectado, a região da imagem que esse objeto detectado consta e outras informações úteis para os protótipos desenvolvidos. Portanto, para o protótipo, como o objetivo era pegar somente toda a área do objeto detectado, foi utilizado como modelo o yolov8x-seg que se propõe a utilizar a segmentação de objetos, ademais, como chamada de retorno foi utilizado como primeiro parâmetro “on_predict_batch_start”, portanto essa chamada de retorno de predição será acionada a cada detecção que o YOLO efetua. Desta forma, era possível obter informações de qual objeto foi detectado além de toda a área de sua silhueta, agora bastaria criar um método responsável por recortar o objeto detectado da cena e demonstrá-lo a cada detecção para o usuário.

5. Transferência neural de estilo: foi implementada a transferência neural de estilo com base no código de (CHOUDHARY, 2020) que forneceu uma base sólida para a criação de uma técnica de transferência de estilo altamente eficaz ao projeto. Contudo, o código da forma em que estava não era útil ao que foi proposto como protótipo, portanto, foi modificado o método responsável pela transferência neural de estilo, fazendo com que ao invés de aceitar o caminho para uma imagem, que seria a imagem a ter o estilo modificado, agora o método receberia uma imagem diretamente como parâmetro.
6. Integração: Com o sistema de detecção de objeto e o algoritmo de transferência neural de estilo funcionais, foi necessário a integração entre as soluções. A integração visa retornar a segmentação do objeto que o código feito com o YOLO é responsável por gerar e mandá-la como parâmetro para a função responsável por fazer a transferência neural de estilo. Nessa função, a segmentação do objeto será a imagem que receberá o método da transferência neural de estilo modificada no passo anterior. Em seguida, o método modificado do código de (CHOUDHARY, 2020) retorna o objeto com as características da imagem artística alteradas para o método adicionado como método da chamada de retorno criada no passo 4, porém ainda foi necessário fazer a junção do retorno contendo o objeto com seu estilo artístico modificado com o resto da imagem original. Após isso o resultado é retornado ao usuário, sendo assim concluído o desenvolvimento dos protótipos.
7. Validação de desempenho em diferentes cenários: dispondo de diferentes hardwares, baterias de testes foram efetuadas para se ter a noção de como o sistema desenvolvido

se comportou em cada um desses hardwares, propondo melhorias e configurações para uma execução eficiente e otimizada da solução. Ao realizar essa validação de desempenho em diferentes cenários, houve uma noção clara de como o sistema se comportou em cada hardware utilizado para os testes, além de identificar oportunidades para aprimorar a execução da solução. Os hardwares utilizados são demonstrados pela figura 13

Figura 13 – Configuração de hardware dos dispositivos para testes utilizados

Dispositivos	Computador	Notebook
Processador	Ryzen 5 5600	i5 10500H
Memória Ram	16 GB	16 GB
Placa gráfica	RTX 2060 6GB GDDR6	GTX 1650 4GB GDDR6
Sistema operacional	Windows 10	Ubuntu 20.06

Fonte: Autor

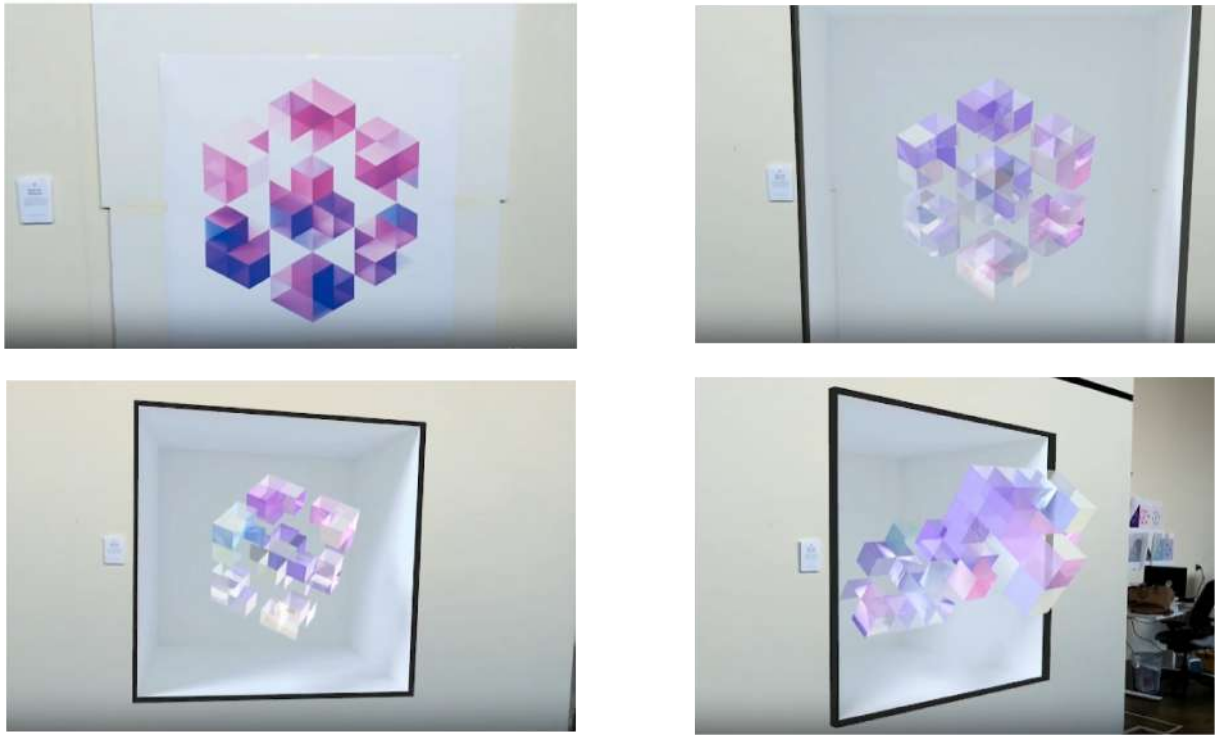
4 RESULTADOS

Nesse capítulo serão detalhados os resultados dos protótipos propostos e elaborados, demonstrando pontos positivos e negativos deles.

4.1 ARCore - Imagem aumentada

Como explicado na seção 2.3, ARCore é um kit de desenvolvimento de software focado em realidade aumentada e desenvolvido pelo Google. Ademais, esse kit de desenvolvimento possui diversos recursos para soluções de realidade aumentada, dentre eles, existe o recurso de imagens aumentadas que permite criar aplicativos de RA que podem detectar e aumentar imagens 2D no ambiente do usuário (GOOGLE, 2023a). Aumentar uma imagem, nesse contexto, seria pôr outro recurso visual digital acima da imagem detectada, também chamada de imagem de referência (ver figura 14).

Figura 14 – Funcionamento do recurso de imagens aumentadas



Fonte: (GOOGLE, 2023a)

Portanto, utilizando essa solução teríamos formas de detectar um objeto em cena, modificar esse objeto para um estilo escolhido e retornar para o usuário, tudo isso em tempo de execução e pelo *smartphone*. O que de certa forma é o objetivo da solução proposta.

Como primeiro protótipo e desafio para entender melhor o funcionamento em termos

de código do recurso de imagem aumentada, foi proposto aplicar opacidade no objeto detectado. Para isso, foi utilizado como base o código disponibilizado no tutorial de CodeLab do autor (ZHOU, 2021). O código utilizado simplesmente aplicava quatro imagens nos extremos diagonais da imagem de referência (o globo terrestre), aplicando uma moldura a ela (ver figura 15).

Figura 15 – Exemplo da imagem aumentada do código base disponibilizado

Fonte: (ZHOU, 2021)

Para ser utilizada como imagem de referência, as imagens a serem detectadas precisam estar no banco de dados de imagens, que podem ser adicionadas em tempo de execução, ou ser utilizado um arquivo de banco de dados (.imgdb) com imagens locais de um diretório específico. As imagens do banco são limitadas a mil e a solução só permite detectar vinte imagens ao mesmo tempo (GOOGLE, 2023a).

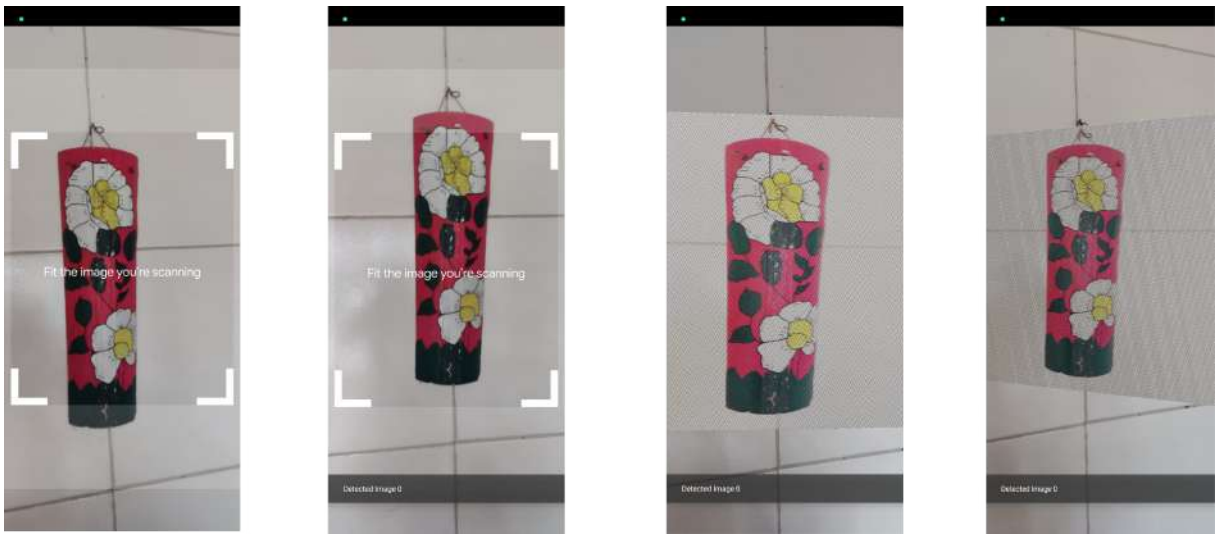
A aplicação permitiu uma série de configurações para a imagem, como alterar opacidade, as cores e tamanho da imagem aumentada. Outro ponto interessante é a configuração

do material, que pôde ser desde a intensidade da luz ambiente até o brilho e reflexo no material que compõe a imagem aumentada. É importante salientar que a imagem aumentada é composta por dois elementos que são um modelo geométrico tridimensional e uma imagem que será a textura que irá envolver esse modelo geométrico.

Para o protótipo, foi utilizado uma arte feita em uma telha como imagem de referência, além disso, como modelo geométrico foi utilizado um objeto quadrangular de pouca espessura e a própria imagem da telha como imagem aumentada. Contudo, em tempo de execução, a imagem da telha era alterada, diminuindo assim a sua opacidade (ver figura 16). Ademais, A escolha de uma telha como imagem de referência se deve aos requisitos e limitações da funcionalidade, como, por exemplo, não ser uma imagem com muitos recursos geométricos e não ter padrões repetitivos, pois imagens com essas características podem ocasionar problemas na detecção e no rastreamento (GOOGLE, 2023a).

Um ponto importante, que foi de grande ajuda, foi a utilização de âncora, a qual é uma funcionalidade que mapeia a localização da imagem de referência e aplica a imagem aumentada com base nessa localização. Essa funcionalidade não é tão interessante de ser usada caso a imagem de referência se mova constantemente, porque isso comprometeria a eficiência da funcionalidade em estimar a localização exata da imagem de referência. Além disso, a capacidade da ferramenta de mensurar o tamanho da imagem referência era essencial para aplicar a imagem aumentada por cima, também era possível definir um tamanho manualmente, mas caso não o fosse feito a ferramenta mensurava automaticamente com o tempo.

Figura 16 – Resultado da imagem aumentada aplicando diminuição da opacidade, utilizando ARCore



Fonte: O autor

4.1.1 Pontos positivos

A facilidade encontrada em reconhecer um objeto e logo após poder adicionar outro recurso visual por cima, foi o ponto crucial para ter sido escolhido como ferramenta de elaboração do primeiro protótipo. Outro ponto positivo é o fato da ferramenta ser do Google, uma das maiores empresas de tecnologia da atualidade, contribuindo para uma ferramenta bem estruturada e documentada, facilitando o uso por parte dos desenvolvedores que a utilizará. Ademais, a ferramenta é compatível com o ecossistema Google, dando vasta capacidade de escalabilidade.

Além disso, o ARCore é uma ferramenta para dispositivos móveis, permitindo que a solução que o utilize seja funcional em qualquer dispositivo, seja ele com o sistema operacional Android ou IOS. Além de que, a utilização dos sensores e outras funcionalidades do *smartphone* melhora a experiência da aplicação.

4.1.2 Pontos negativos

Apesar de possuir várias características necessárias para a elaboração do protótipo proposto como a capacidade de identificar e modificar a imagem de referência para posteriormente ser adicionada como imagem aumentada, a ferramenta de imagens aumentadas do ARCore, possui diversas limitações que aumentaria a complexidade da solução drasticamente.

Dentre as problemáticas presentes, o fato da imagem referência necessitar de certas condições para ser propícia à detecção era a principal causa limitadora. As imagens de referências precisavam que:

1. Preenchesse pelo menos 25% do frame da câmera para ser detectado inicialmente.
2. Fosse plano (por exemplo, não amarrado ou envolto em uma garrafa).
3. Não fosse visualizado em um ângulo altamente oblíquo.

Ademais, outros pontos recomendados também limitariam as imagens de referências, como imagens com resolução de pelo menos 300×300 píxeis e índice de qualidade de pelo menos 75. Índice de qualidade se refere as características da imagem para a ferramenta ser possível melhor detectá-la. Esse índice vai de zero a cem e para ter a noção de qual seria o índice da imagem, seria necessário utilizar a ferramenta chamada arcoreimg incluída no ARCore.

Além de todos esses pontos, o fato da imagem aumentada exigir um modelo geométrico tridimensional, aumentaria ainda mais a complexidade, pois se for utilizado objetos de diferentes formas teria que mudar esse modelo geométrico conforme a forma do objeto, além de

ter que adicionar em tempo de execução o objeto almejado para detecção. O processamento para isso seria lento, pois precisaria de um código de detecção de objeto para, além de obviamente detectar o objeto, salvar a sua forma e sua imagem, além de adicioná-lo ao banco de imagem, levando cerca de 30 ms para cada adição e cada imagem ocupa cerca de 6 KB. Contudo, era também necessário evitar que o banco possua muitas imagens, porque há impacto no desempenho do sistema devido ao maior uso da CPU (GOOGLE, 2023a). E vale lembrar que mesmo após todo esse fluxo a imagem aumentada teria que ser processada para que ocorresse uma transferência neural de estilo, contribuindo para que, mesmo que funcional, o sistema funcionasse bem lentamente.

4.2 YOLOv8 - Detecção de objeto

Após entendermos as limitações presentes na tecnologia abordada anteriormente, partimos para a elaboração do protótipo a partir da ferramenta de detecção de objeto YOLO na sua oitava e última versão até então. O YOLO é uma ferramenta de estado da arte para detecção de objetos, dentre suas principais características a velocidade de detecção destaca-se, permitindo que a aplicação além de detectar objeto possa também fazer todas as outras funcionalidades propostas, dado a margem de tempo após a detecção, não afetando tanto a quantidade de quadros por segundo e tendo uma aplicação suave dentro do possível.

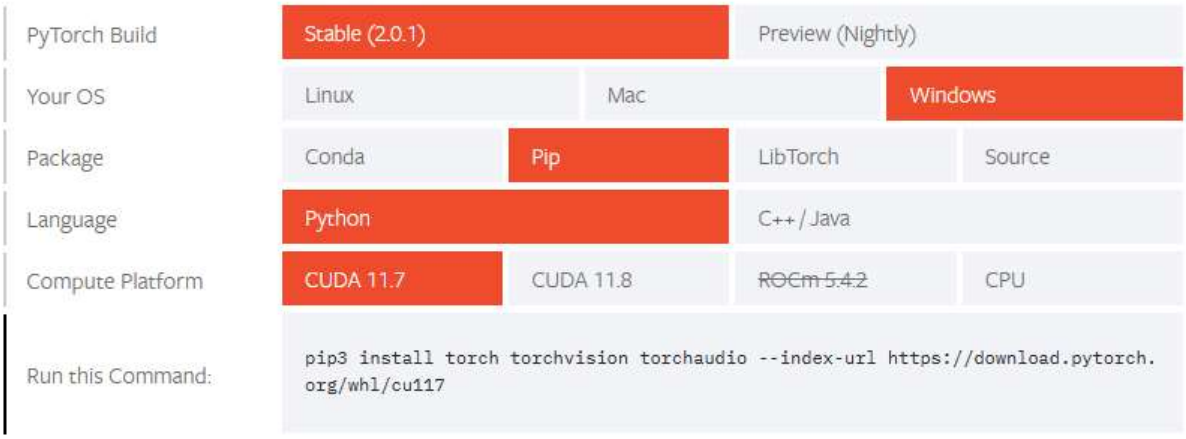
Para funcionamento do YOLO precisaríamos de uma máquina que contenha a linguagem de programação Python em sua versão 3.8 ou superior, além de um ambiente de desenvolvimento com PyTorch na versão 1.7, ou superior. O YOLO pode ser utilizado por uma interface de linha de comando ou por meio de um arquivo Python (.py).

PyTorch é uma biblioteca otimizada de tensores para aprendizado profundo (*deep learning*) que utiliza a capacidade da placa gráfica e do processador (LINUX, 2023a). Tensores são matrizes multidimensionais contendo elementos de um único tipo. A utilização de tensores se torna importante para a atividade de detecção de objeto, ao permitir utilizar esses dados das matrizes de forma rápida e eficiente com o auxílio da placa gráfica.

Vale salientar que é importante a utilização da placa gráfica para processamento de seu código, por tender a ser mais rápida que o processador. Portanto, para que na instalação do PyTorch não ocorram problemas de compatibilidade, é interessante ser feito a instalação pela ferramenta *Start Locally* (LINUX, 2023b), que necessita o preenchimento das informações do seu ambiente de desenvolvimento como o sistema operacional, o gerenciador de pacote que

deseja fazer a instalação, a linguagem de programação escolhida, etc. (ver figura 17).

Figura 17 – Ferramenta start locally, que serve para evitar problemas de compatibilidade entre as bibliotecas que o PyTorch necessita



Fonte: (LINUX, 2023b)

Após preencher tais informações, será mostrado um comando que deverá ser rodado na sua interface de linha de comando desejada. Enfrentamos problema de desempenho antes da utilização dessa ferramenta, pois, mesmo com a instalação do PyTorch por outros meios, ao invés da utilização da placa gráfica estava sendo utilizado o processador da máquina.

A utilização do YOLO exige a definição de um modelo de detecção, que pode ser um customizado, no qual você mesmo treina e define um conjunto de dados para esse treinamento ou utiliza algum modelo pré-treinado de detecção que a própria ferramenta disponibiliza. A ferramenta, na sua oitava versão, disponibiliza cinco modelos distintos: YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l e YOLOv8x. O último caractere do nome do modelo demonstra o tamanho dele, ou seja, o primeiro modelo citado refere-se a nano que é o mais leve e a detecção é mais rápida, porém a capacidade de detectar não é a mais precisa. Contudo, o último citado, o qual é o extra-large, é o mais pesado e possui a detecção mais lenta, entretanto esse é o modelo mais preciso.

Para o nosso projeto, utilizamos os modelos pré-treinados disponibilizados, contudo, não será utilizado modelo de detecção de objeto, mas o modelo de segmentação de objeto. O que difere o nome do modelo de segmentação para o de detecção é simplesmente a adição de -seg após o nome do modelo, então caso utilizemos o modelo nano, para a segmentação o modelo será chamado de YOLOv8n-seg.

O modelo pré-treinado pode detectar 80 classes de objetos distintas, entre elas pessoa, carro, celular, gato, cachorro, etc. Após definir o modelo de treinamento, devemos utilizar o

método *predict* no modelo, esse método exige alguns parâmetros, como *source* que seria a fonte dos dados visuais onde a ferramenta irá fazer a detecção, *show* que seria a opção de mostrar uma interface demonstrando a detecção em tempo real, *classes* que recebe um array com os números inteiros que representam os identificadores das classes dos objetos a serem detectados pelo modelo escolhido, filtrando assim somente pelos identificadores das classes que estão nesse array, dentre vários outros parâmetros.

Além do método *predict*, para o nosso caso foi necessário a utilização do método *add_callback*. Esse método é responsável por executar uma função que o acionamento será em alguma parte do ciclo de detecção do objeto, ou seja, posso ter uma função que irá executar somente quando o método *predict* é finalizado, por exemplo. Mas para o nosso caso, o método *add_callback* precisaria ser acionado a cada início de detecção de objeto, tendo em vista que precisaremos modificar cada frame dos objetos detectados aplicando uma diminuição de opacidade neles para esse primeiro protótipo.

Portanto, com a detecção de objeto funcionando, é necessário algumas alterações que serão feitas no método adicionado ao *add_callback*, a principal é a utilização do OpenCV para visualização das detecções em tempo real, já que não utilizaremos a visualização nativa do YOLO, pois ele é capaz somente de demonstrar o objeto detectado sem as alterações que cogitamos ter nos protótipos, em seu lugar será utilizado uma visualização criada por nós no método adicionado ao *add_callback*. É importante salientar que esse método recebe como parâmetro um objeto de classe que contém informações da imagem tratada. Com acesso a esse objeto, podemos demonstrar a imagem original, por meio do OpenCV, e ainda ter acesso a máscara e outras informações da imagem detectada.

OpenCV é uma biblioteca focada em visão computacional, contendo diversas ferramentas para manipular e processar as imagens digitais. Ela será importante, pois poderemos fazer a junção da imagem original com a imagem do objeto detectado (ver figura 18). Após termos em mãos a imagem original e os objetos em cena, poderemos manipulá-las juntando tais imagens e aplicando a diminuição de opacidade, como visto na figura 19 e com isso teremos o primeiro protótipo funcional. O fluxo de desenvolvimento é resumido pela figura 20, portanto, o que caracteriza o protótipo 1 é a capacidade do código de reconhecer algum objeto em cena, manipular esse objeto detectado diminuindo sua opacidade e depois recolocando o objeto em cena, porém no mesmo local em que foi detectado anteriormente.

É importante ressaltar que, para as figuras utilizadas nos exemplos de protótipo, foi

utilizado um vídeo para melhor representação visual. Portanto, para as figuras de exemplos dos resultados, foi utilizado o vídeo de nome “Cars and Mopeds on Taksin Bridge” disponibilizado no site www.videvo.net [1], a escolha do vídeo se deve por ter objetos em cenas bem definidos, que, além de serem objetos que o modelo de detecção utilizado consegue detectar, o vídeo está em alta resolução, mais precisamente em Full HD (1920×1080). Ademais, o vídeo consta de 14 segundos e demonstra carros, motos e pessoas trafegando. A velocidade de reprodução do vídeo é lenta. Contudo, nada impede dos protótipos serem utilizados por meio de um recurso de obtenção de imagens em tempo real, como uma webcam ou qualquer outro dispositivo conectado ao computador que esteja executando o código do protótipo e consiga captar imagens em vídeo.

Com tais informações e a liberdade que as bibliotecas do Python permitem ter, sobretudo o OpenCV, podemos fazer o primeiro protótipo proposto quando estávamos utilizando o ARCore, o qual é diminuir a opacidade da imagem detectada, mantendo o resto da imagem original. No método adicionado ao *add_callback*, devemos manipular as máscaras dos objetos segmentados para que elas sejam adicionadas na imagem original, mas com a opacidade diminuída, para isso utilizaremos o OpenCV.

Figura 18 – Imagem original a esquerda e a máscara dos objetos detectados a direita

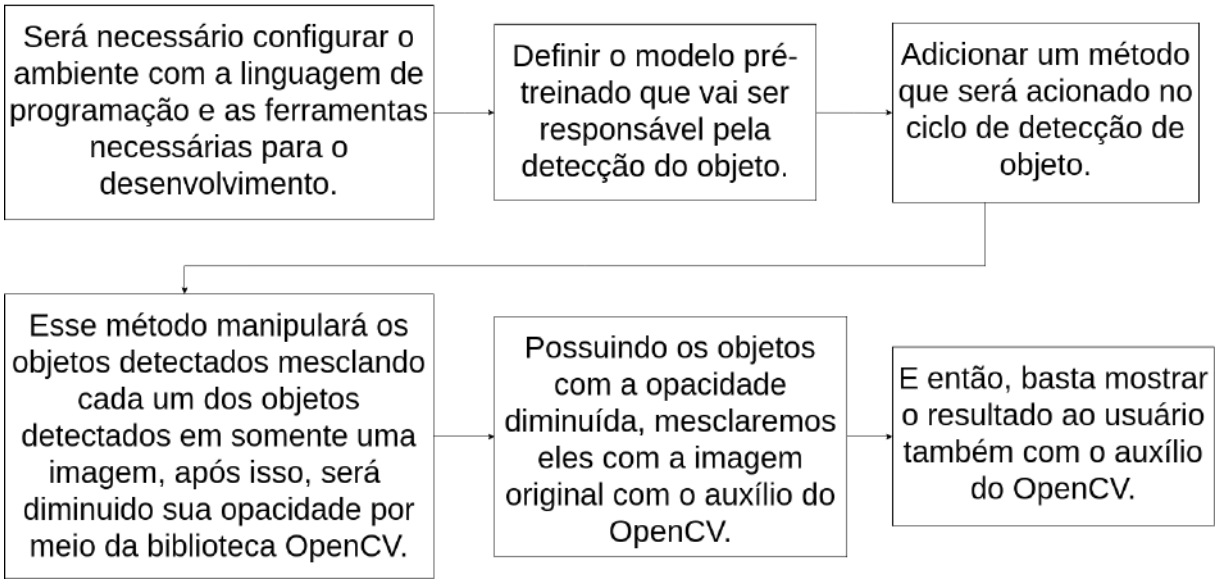
Fonte: www.videvo.net

[1] Videvo é uma plataforma online que disponibiliza uma vasta coleção de clipes de vídeo gratuitos e pagos.

Figura 19 – Conclusão do primeiro protótipo com a imagem original a esquerda e na direita a imagem processada dos objetos detectados com opacidade diminuída e inseridas na imagem original

Fonte: www.videvo.net

Figura 20 – Fluxo de desenvolvimento para o protótipo 1



Fonte: O autor

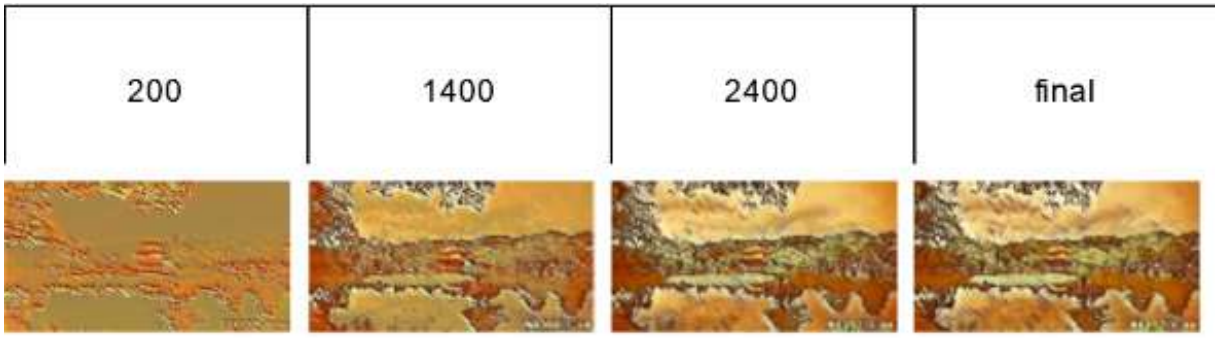
4.3 Transferência Neural de Estilo

Após estar familiarizado com o YOLO e conseguir manipular a imagem detectada, focamos na transferência neural de estilo, que deverá ocorrer somente na imagem detectada. Para

isso utilizaremos como base o código de (CHOUDHARY, 2020), com o nome de *fast neural style transfer*. O código dele já permite a transferência neural de estilo e o treinamento do modelo. Esse treinamento resultará na geração de um arquivo que será utilizado posteriormente para a escolha do estilo neural a qual será feito a transferência neural de estilo. Para o treinamento é possível configurar a quantidade de vezes que o conjunto de dados será treinado em loop. Isso é importante, pois para cada loop o estilo será refinado, como demonstra (CHOUDHARY, 2020) na figura 21.

A figura 21 demonstra a relação entre a iteração (caracterizado pelo número acima) e o resultado do treinamento (abaixo dos respectivos números de iterações), com isso podemos concluir que com mais iterações o resultado do modelo tem mais detalhes, contudo, após certa quantidade de interações a melhoria no resultado é pouca ou imperceptível como acontece com as iterações com número 2400 e a final.

Figura 21 – Quantidade de iterações no treinamento com o seu respectivo resultado na transferência neural de estilo



Fonte: (CHOUDHARY, 2020)

Contudo, com o código base aplicado e funcional no ambiente de desenvolvimento local, tivemos que fazer algumas alterações para que ele fosse útil para o nosso segundo protótipo, que consta em aplicar um estilo neural em todos os objetos detectados. No método responsável pela transferência neural de estilo do código base, ocorria que, após a transferência de estilo, era salva a imagem processada, porém, para o nosso protótipo não era necessária essa funcionalidade, pois seria gasto de processamento no ato de salvar tal imagem e após utilizá-la no nosso sistema de detecção de objeto, além de que isso teria que ser feito para cada frame que ocorresse uma detecção, podendo chegar a 86 detecções por segundo como explicaremos mais a frente.

Portanto, a nossa alteração de código focou somente no método de transferência neural de estilo, mais especificamente em duas funcionalidades, na qual a primeira seria na funcionalidade de salvar a imagem, tendo em vista que isso não seria preciso, e a segunda na

forma com que o método recebia a imagem que seria processada, pois antes o método esperava um caminho para uma imagem local. Então, agora o código recebe um arquivo de imagem diretamente e ao invés de salvar localmente ele retorna essa imagem processada que será utilizada no código de detecção de objeto criado anteriormente.

4.4 Integração do código de detecção com o código de transferência neural de estilo

Para a realização do segundo protótipo, teremos que elaborar a integração dos códigos utilizados anteriormente, para isso é necessário importar o código de transferência neural de estilo no código de detecção de objeto. Após a importação, devemos focar no código de detecção de objeto e a modificação também será na função adicionada ao *add_callback*. Devemos agora mesclar todas as máscaras de cada um dos objetos detectados iterando entre elas e utilizando OpenCV para efetuar essa mesclagem. No final da iteração, será feita a chamada do método que alteramos no projeto responsável pela transferência neural de estilo. Esse método receberá as máscaras mescladas formando somente uma imagem, essa imagem será recebida pelo método responsável pela transferência neural de estilo junto do caminho do modelo de checkpoint, que por sua vez foi o arquivo que resultou do treinamento do estilo neural feito anteriormente e com ele será possível definir o estilo desejado. O método que receberá a imagem da junção das máscaras dos objetos detectados e o caminho do checkpoint model retornará a própria imagem recebida, mas tendo sido processada pela transferência neural de estilo.

A imagem retornada não vem no formato desejado e para efetuar essa mesclagem de duas imagens diferentes o OpenCV exige que elas estejam de mesmo formato, ou seja, como as duas imagens para o sistema são arrays e esses arrays possuem dimensões diferentes, para a mesclagem funcionar é devidamente exigido que os arrays possuam mesma dimensão e essas dimensões são definidas pela resolução da imagem utilizada no sistema de detecção de objeto, contudo em tempo de execução existe a possibilidade de efetuar tais mudanças para as imagens possuírem mesmas dimensões e a possibilidade de efetuar a mesclagem. Portanto, feito a mesclagem dos objetos detectados que passaram pela transferência neural de estilo, basta mesclar agora com a imagem original e mostrar o resultado para o usuário, logo, concluímos o nosso segundo protótipo (ver figura 22).

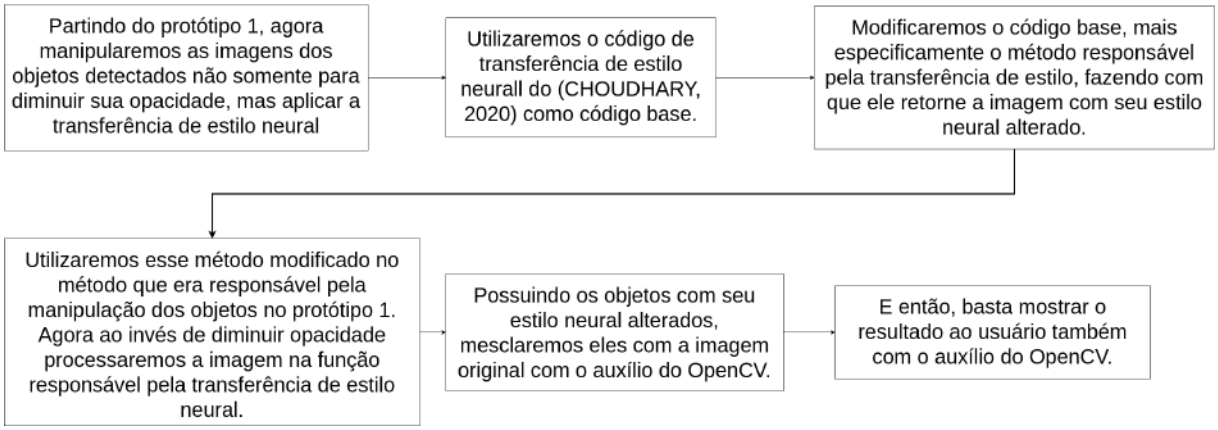
Para resumo dos passos efetuados para o desenvolvimento do segundo protótipo confira a figura 23. Para o protótipo 2, o objetivo era que ele conseguisse detectar os objetos em cenas, efetuar a transferência neural de estilo nesses objetos detectados, adicionar os objetos com

seu estilo neural modificado novamente a cena em seus respectivos locais, porém era aplicado a todos os objetos o mesmo estilo neural.

Figura 22 – Conclusão do segundo protótipo com a imagem original na esquerda e na direita a imagem processada contendo os objetos detectados e com seus estilos neurais alterados

Fonte: www.videvo.net

Figura 23 – Fluxo de desenvolvimento para o protótipo 2



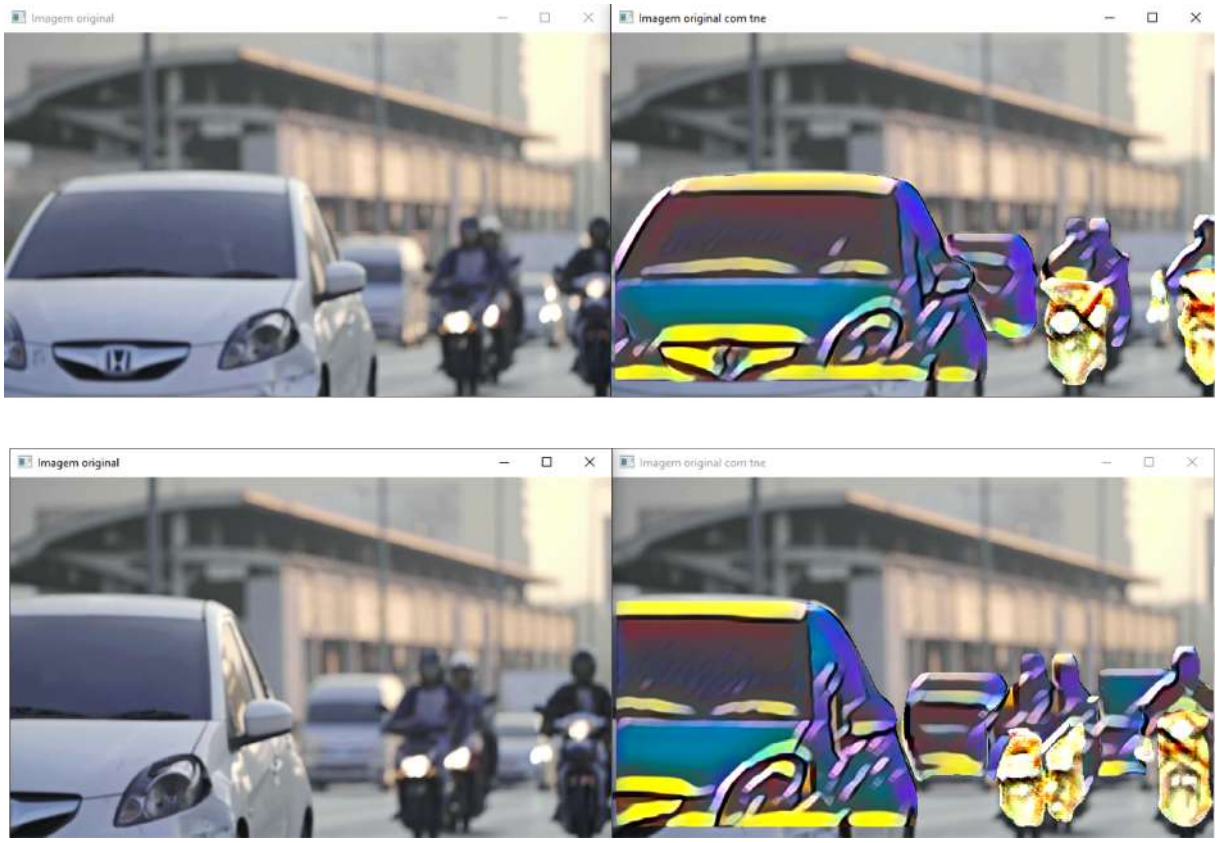
Fonte: O autor

Para o terceiro e final protótipo ocorrerá uma pequena mudança no que foi estabelecido para o segundo, o terceiro terá que aplicar o estilo neural conforme a classe do objeto

detectado, ou seja, se forem detectados quatro pessoas na imagem, todas as quatro pessoas terão o mesmo estilo neural e assim acontecerá para todos as outras classes que o modelo pré-treinado YOLOv8x-seg consegue detectar. Tendo em mente que o que define o estilo neural que será utilizado no método de transferência neural de estilo é o arquivo do checkpoint model, definiremos dois arquivos distintos onde cada um será responsável por um estilo neural, tendo apenas dois estilos possíveis, dividirei as classes entre os identificadores pares e ímpares.

As mudanças necessárias para o protótipo dois também atingirá a necessidade de mudar a lógica de mesclagem entre todos os objetos detectados, agora ao invés de efetuar essa mesclagem, será feito a transferência neural de estilo para cada objeto detectado segundo o seu estilo definido pelo seu identificador de classe. Quando finalizada as transferências de estilo em cada um dos objetos, então nesse momento a mesclagem das máscaras se faz necessária para logo após serem juntadas à imagem original. Com todo esse fluxo feito concluímos o terceiro protótipo (ver figura 24, na qual carro e pessoas compartilham de mesmo estilo neural, enquanto a motocicleta compartilha de um estilo neural diferente).

Figura 24 – Conclusão do terceiro protótipo com a imagem original na esquerda e na direita a imagem processada contendo os objetos detectados com seus estilos neurais alterados, de acordo com o identificador de sua classe



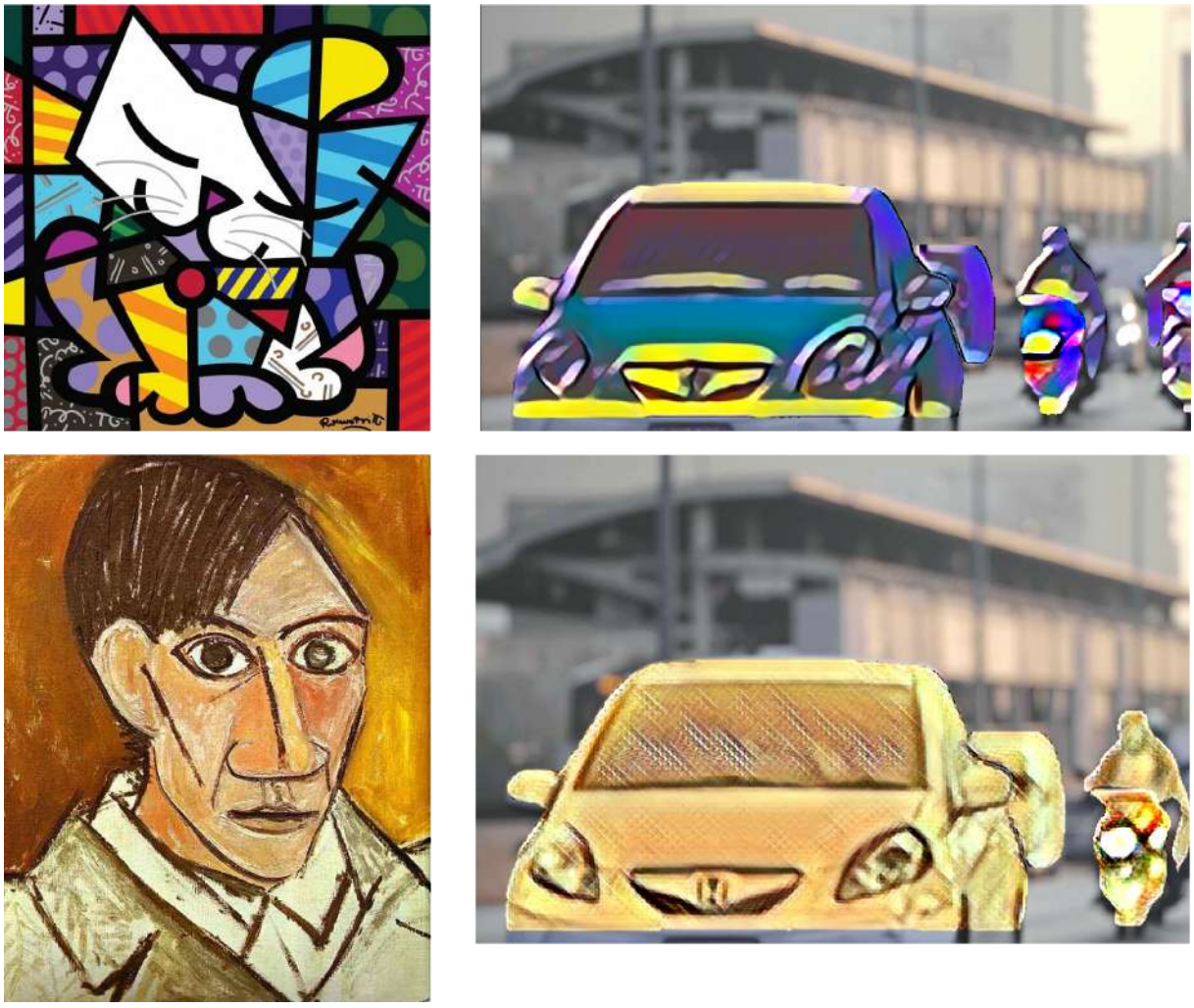
Fonte: www.videvo.net

Vale ressaltar que os estilos neurais utilizados foram:

1. De Pablo Picasso, mais especificamente da sua obra “Autorretrato”.
2. De Romero Britto, da sua obra “Gato”.

No entanto, a escolha dos estilos artísticos se deu por conta da diferença entre as obras (enquanto uma é mais monocromática a outra é mais colorida), além disso, era importante explorar como o protótipo iria se portar tendo que lidar com estilos artísticos tão diferentes. Veja a representação do protótipo aplicando o estilo neural ao lado das obras utilizadas na figura 25.

Figura 25 – As obras que passaram no método de treinamento e foram responsáveis pela característica das imagens que passaram pela transferência neural de estilo



Fonte: www.videvo.net

Contudo, mesmo com o terceiro protótipo funcional, nos casos onde existiam muitos objetos detectados em cena, a lógica que aplica a transferência neural de estilo, que ocorre em cada um dos objetos, consumia muito processamento, ocasionando poucos frames na execução do protótipo. A funcionalidade de transferir o estilo neural é a que mais demanda recursos,

portanto em um cenário onde existissem muitos objetos detectados e para cada um fôssemos efetuar essa funcionalidade, poderíamos chegar a menos de um quadro por segundo em sua execução.

Para solucionar esse problema, mesclaremos todos os objetos que compartilham do mesmo estilo em uma mesma imagem e só depois efetuaremos a transferência de estilo nessa imagem. Tendo em mente que só temos dois estilos diferentes, no pior cenário só teremos que chamar o método responsável pela transferência duas vezes, fazendo com que a média de frames não fique tão instável. Após esse ajuste de desempenho, foi feito um ajuste que permite agora a análise de vídeos do dispositivo local de armazenamento ou disponíveis na internet por meio do *YouTube*.

Contudo, a média de frames por segundos alcançou algo em torno de seis frames utilizando a placa gráfica do computador, o dobro comparado com a primeira versão do protótipo 3, mas permanece com a média muito abaixo do desejado. Há algumas formas de aumentar a desempenho, como, por exemplo, utilizar de uma placa gráfica mais potente, como não dispomos de tal para o trabalho em questão, faremos uma alteração no código responsável pela transferência neural de estilo.

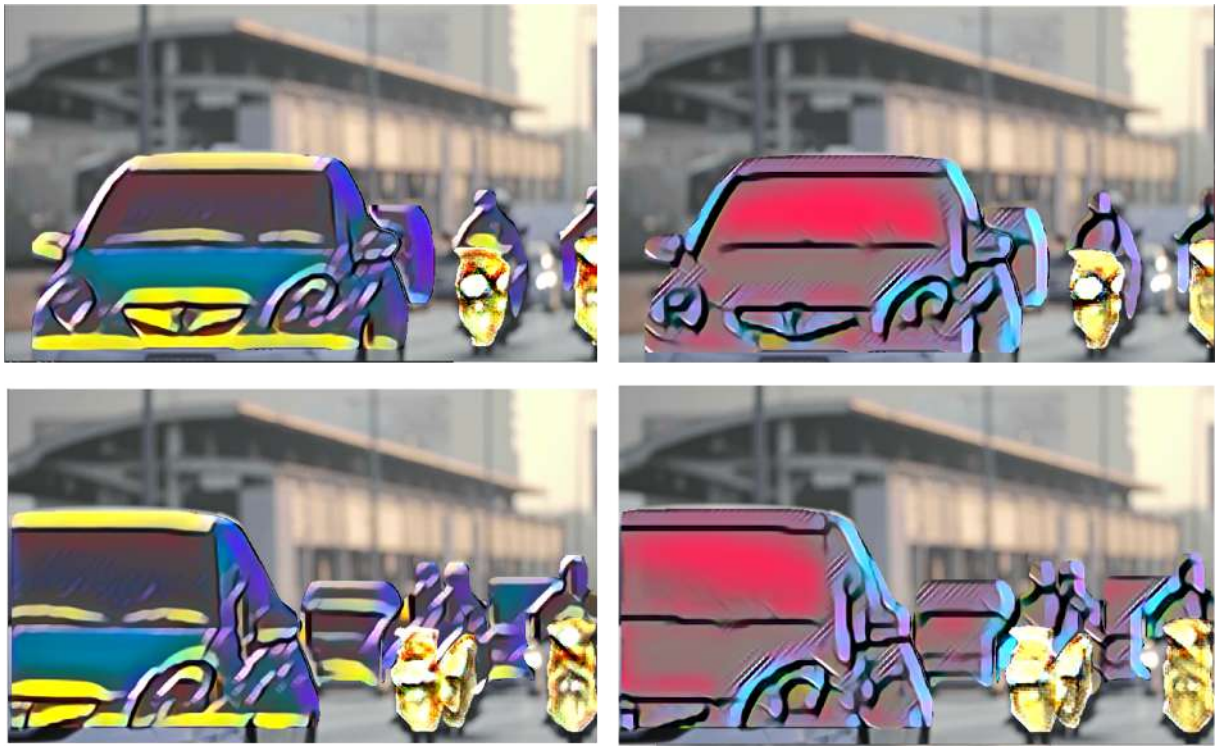
Essa alteração ocorrerá no método responsável pelas camadas convolucionais, nesse método é possível configurar a quantidade de informações que as camadas terão da imagem processada. Portanto, caso configurado para ser diminuída a quantidade de informações presentes em cada camada, ao efetuar o treinamento do estilo desejado, o arquivo do modelo de checkpoint gerado será bem mais leve e a sua utilização não demandará tanto processamento quanto antes, aumentando assim a quantidade de frames do protótipo.

Contudo, modificar a quantidade de informações da imagem no treinamento resultará em um estilo diferente de quando as camadas convolucionais continha mais informações das imagens, isso será exemplificado na figura 26. É notória a diferença entre os estilos, principalmente no estilo de Romero Britto, além de que é perceptível mais detalhes do objeto detectado presentes no estilo do terceiro protótipo original. Esperava-se que, para o estilo de Romero Britto, o protótipo tivesse dificuldade em identificar características do estilo dessa obra, por ser uma obra muito colorida e com vários padrões distintos na textura, dificultando a capacidade do algoritmo de identificar pontos-chave do estilo neural. Por exemplo, nas transferências neurais de estilo da obra de Picasso, no resultado da transferência feita pelo algoritmo é possível identificar uma textura que tenta passar a intenção do objeto ser pintado a mão, isso é possível, pois a

obra de Picasso em quase toda sua extensão transmite esse padrão, facilitando a obtenção dessa característica pelo algoritmo.

Para fins de entendimento do fluxo de desenvolvimento do terceiro protótipo ver figura 27. Por fim, o que caracteriza o terceiro protótipo é a capacidade do código de conseguir detectar objetos em cena a cada frame, aplicar um dos dois estilos neurais escolhidos em cada um dos objetos detectados. Contudo, o que determinará qual estilo neural aplicado será o índice do objeto existente no modelo de detecção previamente treinado pelo conjunto de dados COCO [2], ou seja, os objetos com índices pares serão representados por um estilo neural enquanto os objetos com índices ímpares serão representados por outro estilo neural (no caso da figura 26 as motos possuem o índice impar e ficam com o estilo de Picasso, enquanto os carros e as pessoas ficam com índices pares, portanto, com o estilo de Romero Britto). Esses objetos retirados das cenas e com seus estilos neurais alterados serão juntados e recolocados na cena original para que no final o resultado do frame seja mostrado ao usuário.

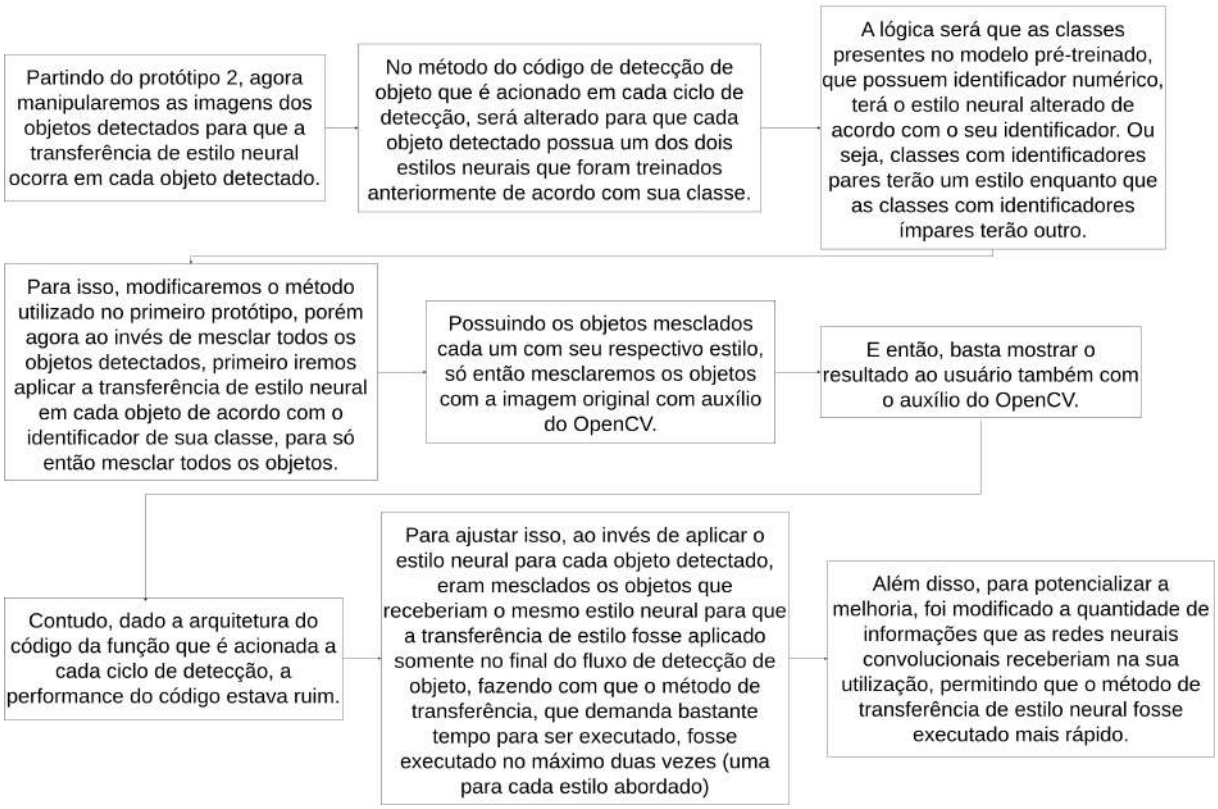
Figura 26 – Terceiro protótipo original (à esquerda) comparado com terceiro protótipo com redução de informações retidas nas camadas das redes neurais convolucionais (à direita)



Fonte: www.videvo.net

[2] O YOLO dispõe dos modelos YOLOV8-seg, modelos pré-treinados por conjuntos de dados, que no caso é o conjunto de dados COCO, capaz de detectar até 80 classes de objetos, na qual cada classe possui um identificador/índice que vai de 0 a 79. Disponível em https://cocodataset.org/#home. Acesso em: 20 de nov. de 2023

Figura 27 – Fluxo de desenvolvimento para o protótipo 3



Fonte: O autor

Contudo, além do vídeo para exemplificar os protótipos, também foram utilizadas imagens de gravação gerada em tempo de execução com o auxílio de uma webcam capaz de gravar em Full HD conectada diretamente no dispositivo no qual o código dos protótipos estavam sendo executados. Demonstrando capacidade do código de ser executado em um vídeo gravado anteriormente ou enquanto ocorrem os processos de detecção e transferência neural de estilo. Ver figura 28.

Figura 28 – Protótipo 1 na esquerda, protótipo 2 no meio e protótipo 3 na direita. Imagens feitas utilizando como dispositivo de entrada uma webcam

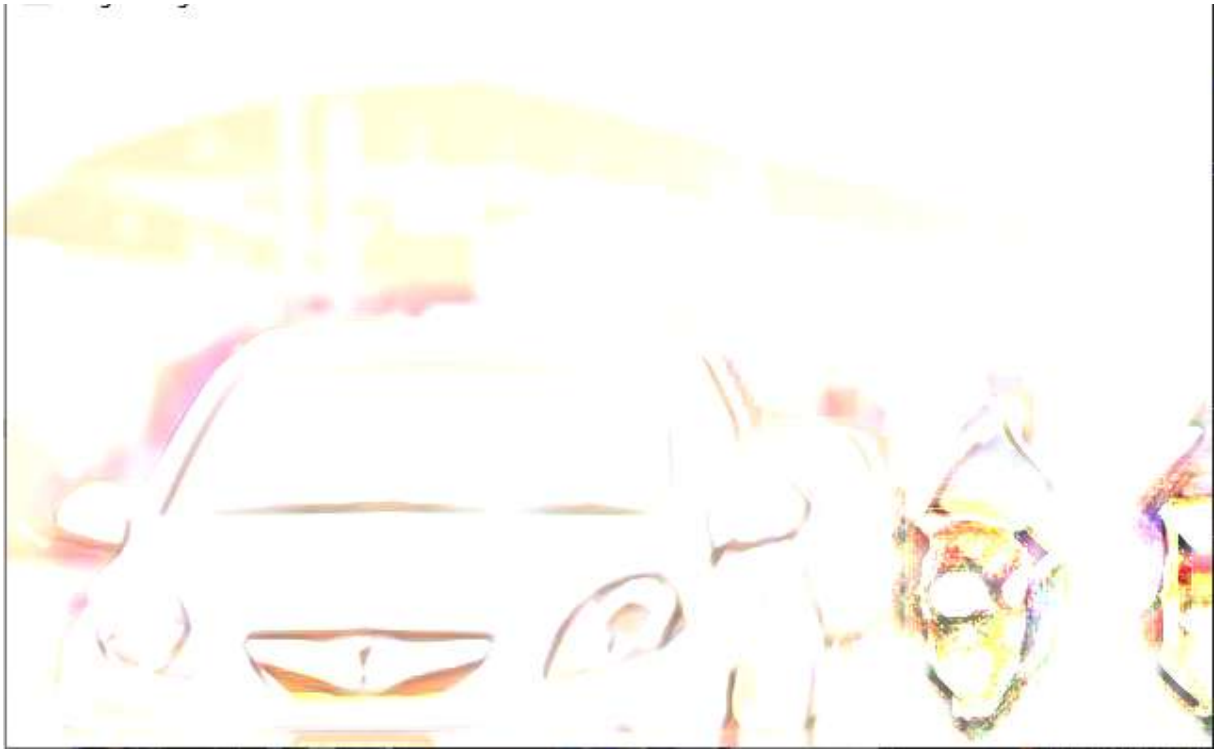
Fonte: O autor

4.5 Falhas enfrentadas

Python em conjunto do OpenCV permite diversas funcionalidades para manipulação de imagens, o que foi de essencial ajuda para a conclusão de nossos protótipos, contudo com uma má manipulação de imagens enfrentaremos alguns problemas.

Na utilização do método responsável pela transferência neural de estilo, mesmo que se cortada a imagem somente no objeto desejado, o estilo era aplicado em toda a dimensão da imagem, ocasionando problemas como da figura 29.

Figura 29 – Problema quando a transferência neural de estilo era processada na imagem toda



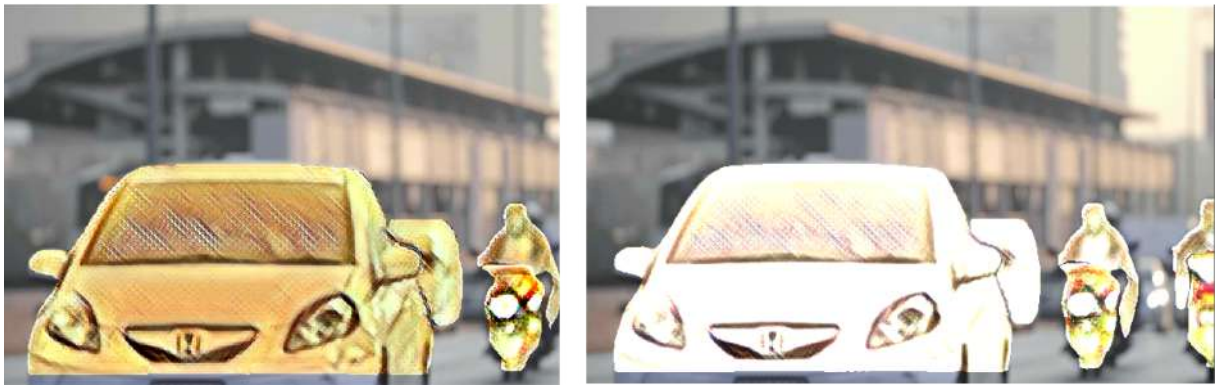
Fonte: www.videvo.net

Para resolução desse problema, era necessário efetuar o corte da imagem após a transferência de estilo, para que somente o objeto permanecesse com essa característica.

Outro problema que ocorreu, que era menos perceptível quanto o primeiro, mas tão inconveniente quanto, era quando ocorria a transferência de estilo, porém a imagem original era sobreposta pelo objeto com o novo estilo (ver figura 30, na qual na esquerda está a imagem correta e na direita o problema relatado).

Para resolver esse problema, no local onde o objeto detectado com seu estilo modificado deveria estar, foi necessário fazer o recorte exatamente no mesmo local na imagem original

Figura 30 – Problema quando a transferência neural de estilo era sobreposta na imagem original



Fonte: www.videvo.net

para que não ficasse anteposta a imagem do objeto com seu estilo modificado. Aplicando essa solução, o problema foi corrigido.

4.6 Performance da solução

Em termos de performance teremos a disposição dois dispositivos diferentes para testes no qual o primeiro é um computador de mesa e outro um notebook e suas configurações de hardware são demonstradas na figura 31.

Figura 31 – Configuração de hardware dos dispositivos para testes

Dispositivos	Computador	Notebook
Processador	Ryzen 5 5600	i5 10500H
Memória Ram	16 GB	16 GB
Placa gráfica	RTX 2060 6GB GDDR6	GTX 1650 4GB GDDR6
Sistema operacional	Windows 10	Ubuntu 20.06

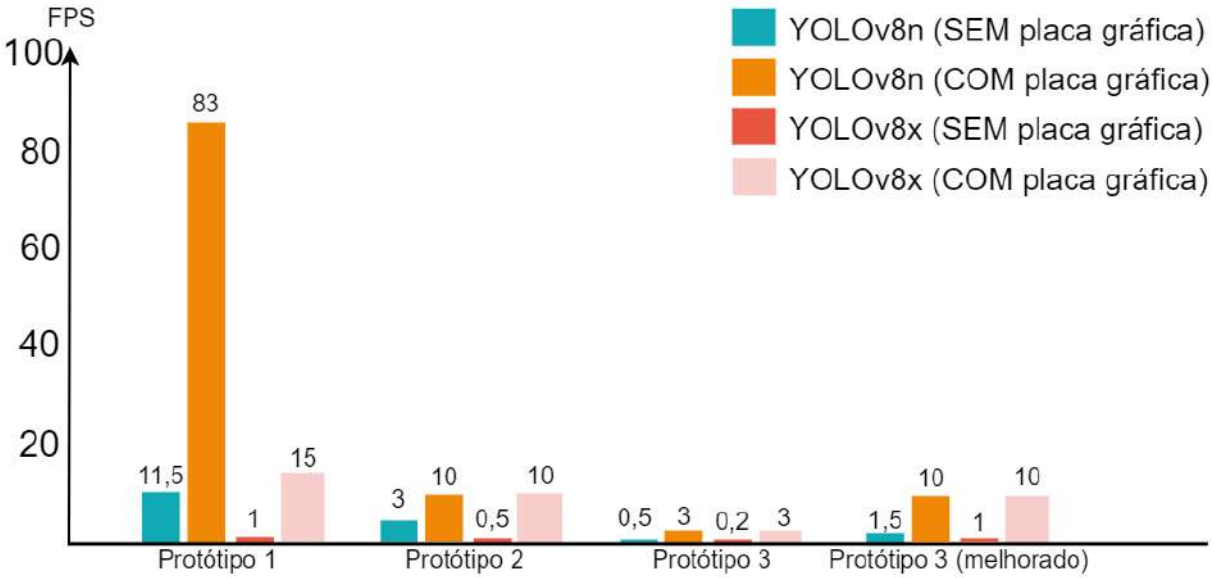
Fonte: Autor

Na figura 32 serão demonstradas as performance do computador na execução dos protótipos em cenários onde os dispositivos utilizam ou não placa gráfica e com dois modelos pré-treinados diferentes para a detecção de objeto (o mais leve e o mais pesado).

Alguns pontos são relevantes na figura 32, como que entre o primeiro e segundo protótipo houve uma grande queda de desempenho, isso se deve especialmente pelo fato de que no segundo protótipo ocorre a utilização do sistema de transferência neural de estilo e ele demanda bastante recurso do dispositivo. Por conta disso, a partir do segundo protótipo a

mudança de modelo traz um ganho irrisório nos cenários em que não é utilizado a placa gráfica, enquanto não há mudança de desempenho entre os modelos quando utilizando a placa gráfica, nesse caso os frames são limitados unicamente pelo método de transferência neural de estilo.

Figura 32 – Performance do computador

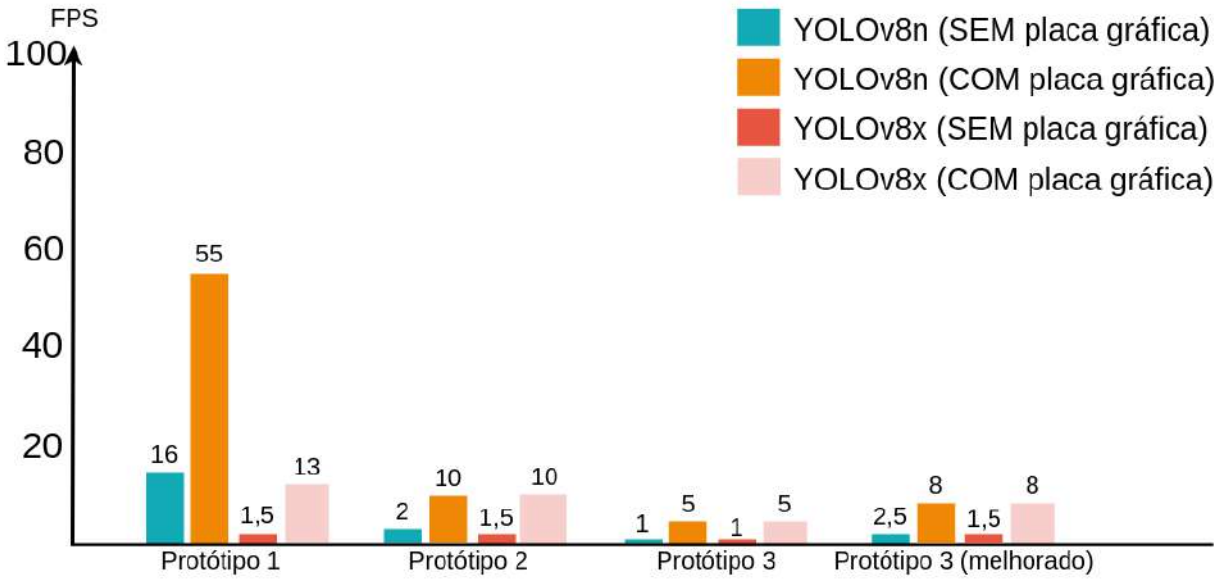


Fonte: Autor

O protótipo 3 perde muito em questão de desempenho e, dependendo do vídeo em que está ocorrendo a análise, pode ser ainda bem pior nesse quesito, contudo há uma melhoria na quantidade de frames no protótipo 3 melhorado, e ainda há a possibilidade de ganho de desempenho quando há apenas objetos que serão modificados pelo mesmo estilo neural.

Contudo, no outro dispositivo utilizado para testes (notebook), o desempenho se alterou um pouco (ver figura 33). Apesar da performance do computador ter maior margem no protótipo 1 com a utilização da placa gráfica, nos outros protótipos ela foi ligeiramente pior, igual, ou até mesmo superior ao computador como no protótipo 3, é importante lembrar que isso pode ter ocorrido dado a instabilidade presente no protótipo 3 original sendo ajustado para a sua versão final. Entretanto, o desempenho sem a utilização da placa gráfica foi inferior em quase todos os cenários, seja no computador ou no notebook, indicando melhor capacidade de processamento por parte das placas gráficas em detrimento dos processadores.

Figura 33 – Performance do Notebook



Fonte: Autor

5 CONCLUSÕES E TRABALHOS FUTUROS

Na elaboração do trabalho em questão, foram levantados diversas tecnologias que poderiam contribuir para a execução dos protótipos propostos. Isso demonstra o grande acervo presente para programas que utilizem de computação visual, e essa quantidade tende a crescer, por conta do interesse dos desenvolvedores e empresas nessa área, dado que haverá crescimento de investimento nela. Outro ponto que favorece essa área de visão computacional é que é constantemente desenvolvido e lançado no mercado de hardware diversos dispositivos com desempenho cada vez mais superior a versões anteriores, permitindo maior liberdade para as soluções, tendo em vista que com maior performance, melhor e mais rápida a solução será executada. Além de que, assim como o hardware, os softwares também evoluem, sendo mais performáticos e eficientes.

Alcançamos os protótipos almejados para o trabalho, além de detalharmos o seu desenvolvimento e execução. E apesar de não termos dispositivos de última geração para a execução e desenvolvimento dos protótipos, o resultado foi satisfatório. Isso contribui para aqueles que dispõem de dispositivos com performance semelhantes e também almejam tentar executar alguma solução parecida, pois terá uma noção aproximada de como funcionará em seu dispositivo e do que pode ser feito com as tecnologias existentes atualmente, contribuindo para áreas como produção de arte digital e reconhecimento de objetos em tempo real.

Dentre os pontos que alcançamos no trabalho, vale destacar a implementação da funcionalidade de imagem aumentada o ARCore, listando os pontos que a tornou uma das tecnologias abordadas e discutindo o porquê dessa tecnologia não ter sido utilizada na versão final do trabalho. Além disso, detalhamos a ferramenta YOLO em sua última versão, implementando-a para a função de detecção de objeto e configurando os modelos pré-treinados que a tecnologia dispõe, para obtermos melhor desempenho no trabalho.

Ademais, o trabalho também utilizou da funcionalidade de transferência neural de estilo em conjunto da detecção de objeto, essa integração entre as duas tecnologias foi a principal contribuição do nosso trabalho, resultando em uma solução única que combina detecção de objeto com a criação de imagens estilizadas com mais de um estilo e em mais de um objeto em cena.

Reconhecemos algumas limitações em nosso estudo. Embora a solução desenvolvida tenha gerado ótimos resultados, é fato que, com os dispositivos de hardwares utilizados, a solução final tenha alcançado pouca média de frames por segundos, apesar das melhorias feitas. Outra

questão também limitante é que tais soluções exigem uma placa gráfica dedicada, que nem todos os computadores pessoais possuem.

Para trabalhos futuros, recomendamos a melhoria na arquitetura do código responsável pela transferência neural de estilo, o qual foi o maior limitante de performance do protótipo final, além de ter uma necessidade de melhoria no treinamento dos estilos para que maiores características do estilo utilizado fosse presente no resultado final. Como melhoria, poderia ser feita a instanciação do modelo de transferência neural de estilo uma única vez e essa mesma instância conseguir fazer a transferência neural de estilo de todas as imagens que ela receber, fazendo isso os protótipos receberiam uma grande melhoria em desempenho. Contudo, com atualizações constantes no YOLO, muito provavelmente haverá novas versões ou melhorias na versão utilizada no trabalho, dando maior margem de desempenho para a solução, além de maior exito na tarefa de detectar objetos. Outro ponto que seria interessante para o trabalho seria a disponibilidade da solução na internet para qualquer um poder testá-lo.

Em suma, este trabalho apresentou uma solução inovadora e desafiadora que combina a eficiência do YOLO com a criatividade da transferência neural de estilo. Apesar das limitações presentes, demonstramos que nosso trabalho alcançou resultados promissores e acreditamos que essa abordagem tem potencial significativo para aplicações diversas, seja na área de desenvolvimento de software até para quem é focado em artes visuais digitais. Caso deseja utilizar o protótipo elaborado segue o link do código disponível no GitHub https://github.com/lcslimas/neural-style-transfer-in-detected-object.

REFERÊNCIAS

AMAZON. **What is a Neural Network?** 2023. https://aws.amazon.com/pt/what-is/neural-network/. Acesso em: 24 Mai. 2023.

APPLE. **Apple Vision Pro - Apple**. 2023. https://www.apple.com/apple-vision-pro/. Acesso em: 21 Jun. 2023.

APPLE. **ARKit: Apple Developer**. 2023. https://developer.apple.com/augmented-reality/arkit/. Acesso em: 14 Mai. 2023.

AZUMA, R. T. Recent advances in augmented reality. **IEEE computer graphics and applications**, IEEE, v. 21, n. 6, p. 34–47, 2001.

BROWNLEE, J. **Deep Learning for Computer Vision: Image Classification, Object Detection, and Face Recognition in Python**. Machine Learning Mastery, 2019. Disponível em: https://books.google.com.br/books?id=DOamDwAAQBAJ.

CARMIGNIANI, J.; FURHT, B. Augmented reality: An overview. In: ____. [*S. l.*: *s. n.*], 2011. p. 3–46. ISBN 978-1-4614-0063-9.

CHEN, Y.; WANG, Q.; CHEN, H.; SONG, X.; TANG, H.; TIAN, M. An overview of augmented reality technology. **Journal of Physics: Conference Series**, IOP Publishing, v. 1237, n. 2, p. 022082, jun 2019. Disponível em: https://dx.doi.org/10.1088/1742-6596/1237/2/022082.

CHOUDHARY, Y. **Fast Neural Style Transfer | Kaggle**. 2020. https://www.kaggle.com/code/yashchoudhary/fast-neural-style-transferl. Acesso em: 26 Jun. 2023.

FURHT B.; LIVINGSTON, M. A. **Handbook of Augmented Reality**. [*S. l.*]: Springer, 2011.

G1. **Pokémon Go para iOS e Android levará monstrinhos para o mundo real**. 2015. Disponível em: https://g1.globo.com/tecnologia/games/noticia/2015/09/pokemon-go-para-ios-e-android-levara-monstrinhos-para-mundo-real.html.

GATYS, L. A.; ECKER, A. S.; BETHGE, M. **A Neural Algorithm of Artistic Style**. 2015.

GOOGLE. **Adicionar dimensão às imagens | ARCore | Google for Developers**. 2023. https://developers.google.com/ar/develop/augmented-images?hl=pt-br. Acesso em: 21 Jun. 2023.

GOOGLE. **O que há de novo no ARCore | Google Developers**. 2023. https://developers.google.com/ar/whatsnew-arcore?hl=pt-br. Acesso em: 15 Mai. 2023.

GOOGLE. **Visão geral do ARCore e de ambientes de desenvolvimento compatíveis | Google Developers**. 2023. https://developers.google.com/ar/develop?hl=pt-br. Acesso em: 15 Mai. 2023.

Grand View Research. **Augmented Reality & Virtual Reality In Healthcare Market Size, Share & Trends Analysis Report By Component (Hardware, Software, Service), By Technology (Augmented Reality, Virtual Reality), By Region, And Segment Forecasts, 2021 – 2028**. 2019. https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-in-healthcare-market. Acesso em: 21 Jun. 2023.

Grand View Research. **Computer Vision Market Size, Share and Trends Analysis Report By Component, By Application (Automotive, Healthcare, Robotics), By Product (PC-Based, Smart Camera), By Region, And Segment Forecasts, 2020-2027**. 2021. https://www.grandviewresearch.com/industry-analysis/computer-vision-market. Acesso em: 03 Mar. 2023.

GRIGG, T. **Concept Learning and Feature Spaces**. 2019. https://towardsdatascience.com/concept-learning-and-feature-spaces-45cee19e49db. Acesso em: 14 Jun. 2023.

GU, J.; DUH, H. Mobile augmented reality game engine. In: ____. [*S. l.*: *s. n.*], 2011. p. 99–122.

HJORTH, L.; RICHARDSON, I. **Pokémon GO: Mobile media play, place-making, and the digital wayfarer**. [*S. l.*]: SAGE Publications Sage UK: London, England, 2017. 3–14 p.

KADISH, D.; RISI, S.; LØVLIE, A. S. Improving object detection in art images using only style transfer. **CoRR**, abs/2102.06529, 2021. Disponível em: https://arxiv.org/abs/2102.06529.

LE, R.; NGUYEN, M.; YAN, W.; NGUYEN, H. Augmented reality and machine learning incorporation using yolov3 and arkit. **Applied Sciences**, v. 11, p. 6006, 06 2021.

LINUX. **PyTorch documentation — PyTorch 2.0 documentation**. 2023. https://pytorch.org/docs/stable/index.html. Acesso em: 25 Jun. 2023.

LINUX. **Start Locally | PyTorch**. 2023. https://pytorch.org/get-started/locally/l. Acesso em: 25 Jun. 2023.

MURPHY, K. P. **Machine learning : a probabilistic perspective**. Cambridge, Mass. [u.a.]: MIT Press, 2013. ISBN 9780262018029 0262018020. Disponível em: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2.

PAELKE, V.; REIMANN, C.; STICHLING, D. Foot-based mobile interaction with games. In: . [*S. l.*: *s. n.*], 2004. v. 74, p. 321–324.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. **You Only Look Once: Unified, Real-Time Object Detection**. 2016.

REDMON, J.; FARHADI, A. **YOLO9000: Better, Faster, Stronger**. 2016. https://pjreddie.com/darknet/yolov2/. Acesso em: 24 Mai. 2023.

SCHMALSTIEG, D.; HöLLERER, T. **Augmented reality: Principles and practice**. [*S. l.*: *s. n.*], 2016. 1-32 p.

SYNOPSYS. **What is LiDAR? Definition, Applications, and Technology**. 2023. https://www.synopsys.com/glossary/what-is-lidar.html. Acesso em: 15 Mai. 2023.

ULTRALYTICS. **Ultralytics YOLOv8 Docs**. 2022. https://docs.ultralytics.com/. Acesso em: 31 Mai. 2023.

ULTRALYTICS. **Pose - Ultralytics YOLOv8 Docs**. 2023. https://docs.ultralytics.com/tasks/pose. Acesso em: 9 Jun. 2023.

VAUGHAN, B.; LAMB, P. **artoolkitx- Github**. 2023. https://github.com/artoolkitx/artoolkitx. Acesso em: 20 Set. 2023.

YUAN, Y. **YOLO Creator Joseph Redmon Stopped CV Research Due to Ethical Concerns**. 2020. https://medium.com/syncedreview/yolo-creator-says-he-stopped-cv-research-due-to-ethical-concerns-b55a291ebb29. Acesso em: 31 Mai. 2023.

ZHOU, K. **Imagens aumentadas do ARCore**. 2021. https://codelabs.developers.google.com/codelabs/augimg-intro?hl=pt-br#0. Acesso em: 21 Jun. 2023.