UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

João Vitor Araujo Porto

Usando Machine Learning para Extração da Semântica de Design de Elementos de Interface de Usuário de Aplicativos do App Inventor

Florianópolis

2024

João Vitor Araujo Porto

Usando Machine Learning para Extração da Semântica de Design de Elementos de Interface de Usuário de Aplicativos do App Inventor

Trabalho de Conclusão de Curso submetido ao curso de Ciências da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Maurício Floriano Galimberti

Florianópolis

2024

Porto, João Vitor Araujo
Usando Machine Learning para Extração da Semântica de Design de Elementos de Interface de Usuário de Aplicativos do App Inventor / João Vitor Araujo Porto ; orientador, Maurício Floriano Galimberti, 2024.
79 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2024.

Inclui referências.

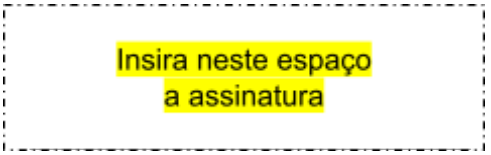
1. Ciências da Computação. 2. Semântica de Design. 3. ; Interface de Usuário. 4. Machine Learning. 5. App Invento. I. Galimberti, Maurício Floriano. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

João Vitor Araujo Porto

Usando Machine Learning para Extração da Semântica de Design de Elementos de Interface de Usuário de Aplicativos do App Inventor

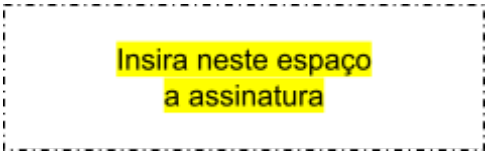
Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel em Ciências da Computação e aprovado em sua forma final pelo Curso Ciências da Computação.

Florianópolis, 12 de dezembro de 2024.



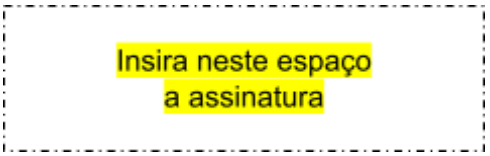
Coordenação do Curso

Banca examinadora



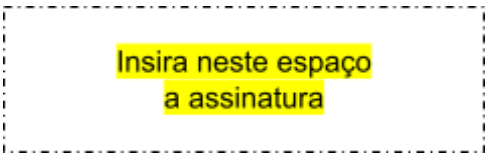
Prof. Dr. Maurício Floriano Galimberti

Orientador



Prof. Dr. Jean Carlo Rossa Hauck

Universidade Federal de Santa Catarina



Prof. Dr. Mauro Roisenberg

Universidade Federal de Santa Catarina

Florianópolis, 2024

AGRADECIMENTOS

Agradeço imensamente à Profa. Christiane Gresse e ao Prof. Aldo von Wangenheim que foram, respectivamente, minha orientadora e coorientador na fase inicial deste trabalho. A Profa. Christiane me deu a oportunidade de entrar em contato com a pesquisa acadêmica e o campo da usabilidade, tendo sido uma excepcional orientadora. O Prof. Aldo me forneceu recomendações primordiais de modelos de Inteligência Artificial.

Agradeço ao Prof. Maurício Galimberti pela generosidade em me orientar na continuidade deste trabalho e aos professores Jean Hauck e Mauro Roisenberg por terem aceitado o convite para participar da banca.

Agradeço aos meus pais, à minha namorada e à minha família por serem incentivadores da conclusão deste trabalho.

E agradeço a Deus por tê-lo concluído.

RESUMO

O ensino de computação na Educação Básica se torna cada vez mais importante, dadas as constantes transformações tecnológicas e sociais pelas quais passa o século XXI. Esse ensino, utilizando ambientes de desenvolvimento como o App Inventor, auxilia os jovens a aprenderem competências básicas de algoritmos e programação. Além da programação em si, outro aspecto fundamental é o *design* das interfaces de usuário (IUs) de aplicativos, que visa maximizar a usabilidade dessas. Desse modo, como parte do processo de ensino-aprendizagem, torna-se importante também a avaliação do *design* de interface dos aplicativos criados pelos alunos. Dentro desse contexto, este trabalho teve por objetivo desenvolver um modelo adotando *machine learning* para a extração da semântica de *design* de elementos de IUs de aplicativos Android criados com App Inventor para posterior integração desse modelo à ferramenta CodeMaster. Para tanto, a execução do trabalho contemplou as etapas de fundamentação teórica, análise do estado da arte, proposição, desenvolvimento e avaliação do modelo. Com isso, obteve-se um modelo de extração da semântica de *design* inédito, colaborando para o aperfeiçoamento futuro da ferramenta CodeMaster, contribuindo para a melhoria do processo de avaliação do *design* de IUs no contexto do ensino de computação na Educação Básica.

Palavras chave: Semântica de Design; Interface de Usuário; Usabilidade; Machine Learning; Deep Learning; App Inventor; Educação Básica.

ABSTRACT

Teaching computing in K-12 education is becoming increasingly important, given the constant technological and social transformations of the 21st century. Using development environments like App Inventor, this education helps young people acquire basic skills in algorithms and programming. Beyond programming itself, another fundamental aspect is the design of user interfaces (UIs) for applications, which aims to maximize usability. Thus, as part of the teaching-learning process, evaluating the interface design of applications created by students also becomes essential. In this context, this work aimed to develop a model using machine learning for extracting the design semantics of UI elements in Android applications created with App Inventor, with the goal of integrating this model into the CodeMaster tool in the future. To accomplish this, the work included stages of theoretical foundation, state-of-the-art analysis, proposal, development, and evaluation of the model. As a result, a unique model for extracting design semantics was developed, enabling the enhancing of the CodeMaster tool and contributing to the improvement of the UI design evaluation process in computing education at the K-12 level.

Keywords: Design Semantics; User Interface; Machine Learning; Deep Learning; App Inventor; K-12 Education.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Para tornar os alunos de hoje cidadãos bem-educados é imprescindível a transmissão de conceitos e técnicas da área da computação, tendo em vista que, no contexto do século XXI, ter domínio restrito ao uso de Tecnologia da Informação não é suficiente (CSTA, 2017). É necessário que os cidadãos sejam capazes de não somente consumir tecnologias, mas também de criá-las ou aperfeiçoá-las de forma inovadora. Assim, torna-se necessária a inclusão do ensino dos conceitos e técnicas de computação no âmbito da Educação Básica.

A computação pode ser ensinada por meio do desenvolvimento de aplicativos para dispositivos móveis, tendo em vista que diversos conceitos relacionados à área como abstração, recursão e iteração, processamento e análise de dados e criação de artefatos reais e virtuais, são empregados ao longo do desenvolvimento dos aplicativos (Grover e Pea, 2013). Além disso, a população, em geral, já possui familiaridade com *smartphones* (Kepios Pte. Ltd., 2019). Nesse contexto, o App Inventor (MIT, 2019) pode ser usado, pois é um ambiente de programação visual baseado em blocos que possibilita a criação de aplicativos Android por qualquer pessoa, inclusive crianças, com ou sem experiência em programação.

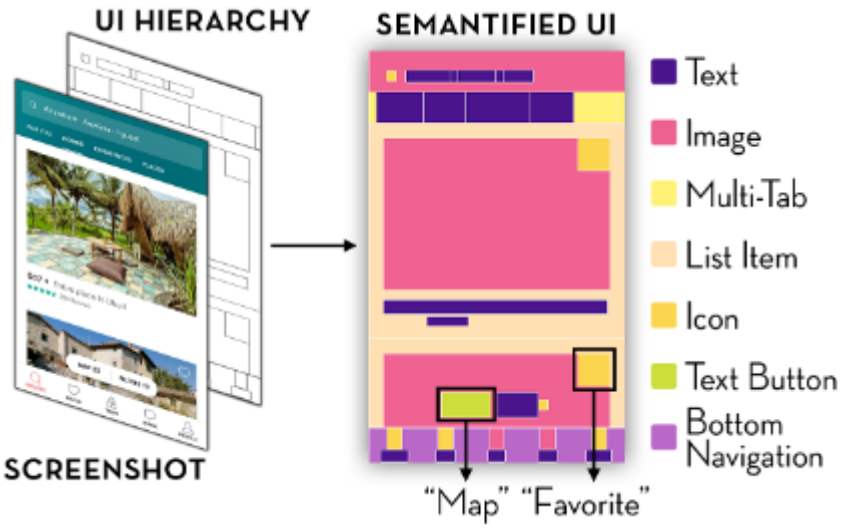
Na medida em que o desenvolvimento de aplicativos não se restringe à programação de suas funcionalidades, mas engloba também o desenvolvimento de suas interfaces de usuário (IUs), o ensino de *design* de interface também é fundamental (Ferreira et al., 2019). O *design* de interface tem por objetivo maximizar a usabilidade e a experiência do usuário, fazendo com que a interação entre usuário e aplicativo seja efetiva, eficiente e satisfatória (Nielsen, 1993). Dessa forma, para proporcionar aos usuários uma interação adequada com os aplicativos, além de uma compreensão ampla do que eles necessitam em termos de funcionalidades, é importante que se desenvolva aplicativos com usabilidade em conformidade com heurísticas, guias de estilo de *design* de IUs etc. Assim, também vem sendo inserido no ensino de computação o ensino de *design* de interface (Ferreira et al., 2019).

Como em toda atividade de ensino, é essencial que os alunos recebam *feedback* instrucional em relação ao *design* de interface criado por eles no desenvolvimento de aplicativos. No entanto, na prática pode ser difícil fornecer um feedback objetivo, consistente e personalizado. Outro obstáculo que dificulta a avaliação efetiva no ensino de computação na educação básica é que a avaliação manual de aplicativos requer tempo e um esforço substancial, o que pode inviabilizar a escalabilidade do ensino de computação a um grande número de alunos (Eseryel et al., 2013). Adicionalmente, devido à escassez de professores de computação na educação básica (Grover et al., 2014), muitos professores de outras áreas introduzem a computação de forma interdisciplinar em suas turmas, enfrentando desafios na avaliação, pois eles não necessariamente possuem experiência na área de computação ou de *design* (Cateté, Snider e Barnes, 2016). Acrescenta-se a isso o fato da avaliação manual ser suscetível a erros devido a diversas razões como inconsistência, fadiga ou favoritismo (Zen, Iskandar e Linang, 2011). Desse modo, a avaliação automatizada torna-se importante, pois reduz a carga de trabalho dos professores, deixando-os livres para despender mais tempo em outras atividades com os alunos (Ala-Mutka e Järvinen, 2004).

Embora já existam algumas ferramentas automatizadas voltadas à avaliação da aprendizagem de computação por meio da criação de aplicativos com App Inventor, focando em conceitos de algoritmos e programação (Alves et al., 2019), como, por exemplo, o CodeMaster (Wangenheim et al., 2018), a maioria não aborda em detalhes a avaliação de conceitos de *design* de interface. Somente o CodeMaster - UI Design avalia automaticamente aplicativos do App Inventor com base em uma rubrica definindo critérios para avaliar o grau da conformidade do *design* de interface desenvolvido pelo aluno em relação a guias de estilos como Material Design (Google Design, 2019), W3C (2008), entre outros (Porto et al., 2018). Mesmo demonstrando uma boa confiabilidade e validade, identificou-se a necessidade de obtenção de informações relacionadas à semântica de *design* de certos elementos de uma IU para que a avaliação seja mais completa. O tamanho da fonte adequado a ser usado para um dado rótulo, por exemplo, dependerá da semântica desse rótulo: um título deve ter tamanho maior do que um subtítulo, que por sua vez deve ter tamanho maior que o corpo do texto e assim por diante.

Consequentemente, critérios que exigem conhecimento da semântica de *design* dos elementos das IUs, atualmente não conseguem ser avaliados pela ferramenta CodeMaster. Assim, tem-se como principal objetivo deste projeto a criação de um modelo de extração da semântica de elementos de *design* de IUs para tornar viável a avaliação desses critérios conforme (Figura 1). Com esse modelo, pode-se extrair dados da semântica de *design* de elementos das IUs de aplicativos não só em termos de seus elementos de *design* como botões, rótulos, caixas de texto etc, mas também em termos das funcionalidades subjacentes a esses elementos, isto é, aquilo que os elementos podem prover em termos da interação do usuário com o aplicativo.

Figura 1 – Extração da semântica de elementos de *design* de interface



Fonte: (Liu et al., 2018)

Uma forma de fazer a extração da semântica é por meio de um modelo automatizado utilizando técnicas e modelos de *machine learning* como clusterização e redes neurais convolucionais (Liu et al., 2018).

Atualmente já existem pesquisas iniciais nessa área voltadas a aplicativos Android (Liu et al., 2018) (Degaki et al., 2022), no entanto, as particularidades dos aplicativos criados com App Inventor a serem aplicados em um contexto educacional tornam necessário um modelo customizado para esse contexto. Assim, a integração futura desse modelo com a ferramenta de avaliação CodeMaster pode possibilitar a avaliação de aplicativos de uma forma mais completa.

1.2 OBJETIVOS

Objetivo geral

O objetivo geral deste trabalho é desenvolver um modelo de extração da semântica de *design* de elementos de IU de aplicativos Android desenvolvidos com App Inventor para a posterior integração desse modelo à ferramenta CodeMaster, aprimorando a avaliação automatizada do *design* visual de aplicativos criados com App Inventor no contexto do Educação Básica.

Objetivos específicos

O1. Estudar as áreas de *machine learning* e *design* de interface buscando fundamentação teórica, técnicas e métodos computacionais aplicáveis à extração da semântica de *design* de elementos de IU;

O2. Analisar o estado da arte em relação à extração da semântica de *design* de elementos de IU de aplicativos Android;

O3. Desenvolver e testar um modelo baseado em visão computacional utilizando *machine learning* para extração da semântica de *design* de elementos de IU de aplicativos criados com App Inventor;

Premissas e restrições

O trabalho é realizado de acordo com o regulamento vigente do Departamento de Informática e Estatística (INE – UFSC) em relação ao Trabalho de Conclusão de Curso. O modelo tem como foco a extração da semântica de *design* de IUs, não abrangendo outras propriedades dessas. Além disso, são suportadas pelo modelo IUs de aplicativos Android, com maior adequação àquelas de aplicativos desenvolvidos com App Inventor.

1.3 MÉTODO DE PESQUISA

O método de pesquisa aplicada a ser utilizado no trabalho é dividido em etapas de acordo com os objetivos específicos.

Etapa 1 – Fundamentação teórica

Nesta etapa é realizada uma análise da literatura nas áreas de *machine learning* e *design* de interface, visando a identificação dos principais conceitos relevantes no O1. Estudar as áreas de *machine learning* e *design* de interface buscando

fundamentação teórica, técnicas e métodos computacionais aplicáveis à extração da semântica de *design* de elementos de IU;

O2. Analisar o estado da arte em relação à extração da semântica de *design* de elementos de IU de aplicativos Android;

O3. Desenvolver e testar um modelo baseado em visão computacional utilizando *machine learning* para extração da semântica de *design* de elementos de IU de aplicativos criados com App Inventor;

O4. Integrar um modelo de extração semântica ao módulo de avaliação de *design* de interface do CodeMaster.presente trabalho. Na área de m*achine learning* será enfatizado o campo de visão computacional e na área de *design* de interface o campo de IUs de aplicativos móveis.

A1.1 – Analisar a área de *machine learning*, nos campos de visão computacional, *deep learning*, clusterização, entre outros;
A1.2 – Analisar a área de *design* de interface nos campos de IUs de aplicativos móveis, avaliação de usabilidade de aplicativos móveis e guias de estilo para o *design* de aplicativos Android.

Etapa 2 – Estado da arte

Analisa-se o estado da arte em relação à extração da semântica de *design* de elementos de IU de aplicativos Android. Para esta etapa será utilizada a técnica de mapeamento sistemático da literatura (Petersen, 2008).

A2.1 – Definir o protocolo de mapeamento sistemático;
A2.2 – Executar a busca e selecionar artigos relevantes;
A2.3 – Analisar e interpretar as informações extraídas.

Etapa 3 – Desenvolvimento

Desenvolvimento e teste do modelo baseado em visão computacional utilizando m*achine learning* para extração da semântica de *design* de elementos de IU de aplicativos criados com App Inventor. O modelo será desenvolvido seguindo um processo de desenvolvimento de redes neurais (Kierski, 2017) (Shuai, 2017) (Polyzotis et al., 2017).

A3.1 – Levantar e analisar requisitos do modelo;
A3.2 – Coletar dados automatizando a coleta de *screenshots*;

A3.3 – Preparar conjunto de dados coletado;
A3.4 – Selecionar um modelo de rede neural;
A3.5 – Treinar rede neural;
A3.6 – Avaliar rede neural;
A3.7 – Afinar parâmetros da rede neural;
A3.8 – Inferir sobre a rede neural.

Etapa 4 – Avaliação

Nesta etapa é feita a avaliação do modelo para identificar sua adequação ao propósito deste trabalho e a capacidade de integração com a ferramenta CodeMaster.

A4.1 – Avaliar a precisão do modelo;
A4.2 – Avaliar o tempo de resposta do modelo.

1.4 ESTRUTURA DO DOCUMENTO

No capítulo 2 é apresentada a fundamentação teórica dos conceitos relacionados às áreas de *machine learning* e *design* de interface. O capítulo 3 apresenta o estado da arte em relação à detecção de elementos de IUs de aplicativos Android com *machine learning*. No capítulo 4 é apresentado o modelo desenvolvido para a extração de semântica de IUs usando *machine learning*. Por fim, o capítulo 5 apresenta a avaliação do modelo.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 *MACHINE LEARNING*

Machine learning é um campo da inteligência artificial que explora a resolução de problemas por meio do ensino de dispositivos computacionais, tornando-os, com isso, aptos a dar respostas adequadas aos problemas. Esses dispositivos são, a grosso modo, algoritmos que têm por finalidade extrair informações úteis de dados brutos e as representar por meio de algum modelo matemático que possa ser usado para fazer inferências (Goodfellow et al., 2016). Podendo ser aplicado em diferentes tipos de problema, incluindo detecção de *spam*, sistemas de recomendação, marcação em fotos de redes sociais, assistentes pessoais ativados por voz, carros autônomos, *smartphones* com reconhecimento facial, entre outros, esse campo vem se consolidando na indústria de desenvolvimento de *software* como um importante provedor de funcionalidades que anteriormente não poderiam ser alcançadas (Data Science Academy, 2019).

Esta seção explora os principais conceitos envolvidos em *machine learning* e que são particularmente pertinentes ao contexto deste trabalho, começando por uma discussão sobre os conjuntos de dados (seção 2.1.1), dos quais as informações úteis são extraídas, passando pelos tipos de aprendizado (seção 2.1.2), aos quais se pode submeter os dispositivos computacionais, seguindo para as redes neurais artificiais (seção 2.1.3), que são os dispositivos que têm mais destaque nesse campo e encerrando com *deep learning* (seção 2.1.4), um subcampo de *machine learning*.

2.1.1 Conjunto de dados

Como a definição de *machine learning* dada indica, os dados têm um papel central na criação do modelo matemático pelo algoritmo (Data Science Academy, 2019). De fato, os dados são tão importantes quanto os algoritmos usados na resolução dos problemas.

Levando isso em consideração, duas das etapas mais importantes durante o desenvolvimento de uma solução com *machine learning* são a coleta e a preparação dos dados para formar o conjunto de dados do qual o algoritmo extrairá as informações necessárias.

Os dados que compõem esse conjunto variam conforme o domínio do problema que se quer solucionar. Para a detecção de objetos e reconhecimento facial, por exemplo, os dados consistem basicamente em imagens ou vídeos.

2.1.2 Tipos de aprendizado

Os algoritmos de *machine learning* podem ser classificados de acordo com o tipo de aprendizado que apresentam (Data Science Academy, 2019). A seguir os principais tipos de aprendizado são definidos.

Aprendizado supervisionado. No aprendizado supervisionado, o algoritmo recebe ao longo de seu treinamento um conjunto de dados rotulados. Durante o treinamento, o modelo matemático é construído com base nos dados e os rótulos dos dados são usados pelo algoritmo para verificar se o modelo criado está convergindo ou não para uma solução apropriada. Isso significa dizer que se o problema a ser solucionado é identificar a existência de gatos em imagens, por exemplo, será dado ao algoritmo um conjunto de imagens rotuladas, de modo que o rótulo de cada imagem indica se a mesma possui ou não um gato, de modo que o algoritmo pode verificar quando o modelo faz a correta classificação de uma imagem ou não.

As principais áreas em que o aprendizado supervisionado é útil envolvem problemas de classificação (em que o modelo deve prever a quais classes cada dado de entrada pertence) e problemas de regressão (que lidam com dados contínuos).

Aprendizado não-supervisionado. Como nem sempre é possível obter um conjunto de dados rotulado e preparado, e, eventualmente, pode não se saber as respostas às perguntas feitas ao modelo, o aprendizado não-supervisionado pode ser aplicado. Nesse tipo de aprendizado, o conjunto de dados que é passado ao algoritmo não contém um resultado desejado específico ou uma resposta correta, de modo que o algoritmo analisa a estrutura dos dados para identificar padrões que possam ser úteis à resolução do problema.

Aprendizado por reforço. Em termos gerais, no aprendizado por reforço (*reinforcement*) aplica-se a estratégia de destacar e priorizar os passos dados ao longo do aprendizado que se aproximam de uma dada meta. Nesse sentido, é comum que se tenha a abstração de um agente que tenta explorar diversas táticas

para atingir da melhor maneira possível um objetivo específico ou melhorar o desempenho em uma tarefa, e, à medida que o agente executa ações que o aproximam de sua meta, são dadas recompensas a ele, de modo que seja possível prever o melhor próximo passo em direção à meta. As escolhas do agente dependem dos aprendizados provenientes de feedbacks anteriores e da tentativa de usar novas táticas. Assim, o processo de aprendizado se dá de forma iterativa e quanto mais iterações com feedbacks, melhor se torna a estratégia do agente, sendo útil especialmente para o treinamento de robôs.

2.1.3 Redes Neurais Artificiais

Uma Rede Neural Artificial (RNA) (Data Science Academy, 2019) é um modelo computacional que simula um cérebro orgânico tentando adquirir a mesma capacidade que ele tem de aprender (obter conhecimento) a partir da experiência.

De forma análoga, uma rede neural artificial é formada por neurônios artificiais, que, a grosso modo, são componentes que calculam a soma ponderada de várias entradas, aplicam uma função sobre essas entradas e passam o resultado da função adiante. Neste contexto, os impulsos elétricos representam os dados, que são passados de neurônio por neurônio. A ideia de que alguns dados excitam mais ou menos os neurônios também está presente por meio da aplicação de pesos sinápticos, que podem ser positivos ou negativos, como indica a figura 2.

Figura 2 – Multiplicação de matrizes entre sinais de entrada x e pesos sinápticos w

$$\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$$

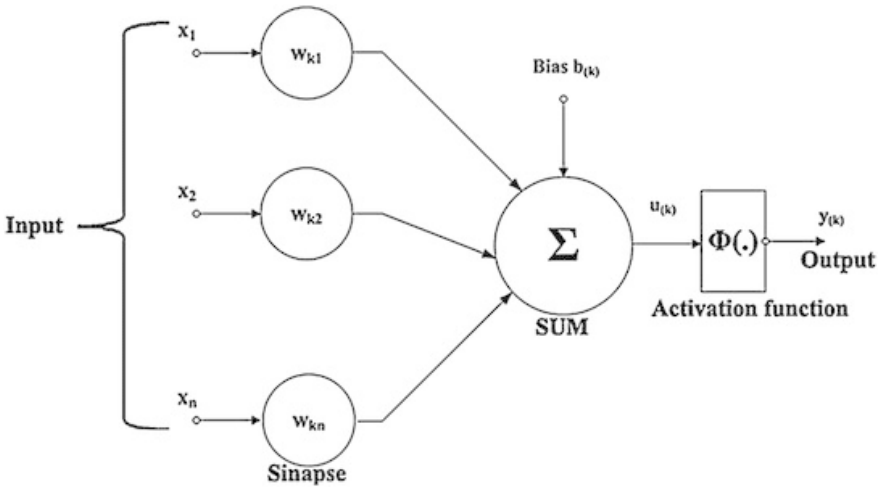
$$\begin{bmatrix} 1 & x_{11} & \dots & x_{1d} \\ 1 & x_{21} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & \dots & x_{nd} \end{bmatrix} \times \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Fonte: (Data Science Academy, 2019)

Na figura 3, os pesos são dados por $w_{k,n}$, em que k é índice do neurônio e n é uma referência ao terminal de entrada da sinapse a qual o peso sináptico se refere. O corpo do neurônio é constituído por duas funções. A primeira é função de combinação, que é o somatório dos estímulos (dados) de entrada multiplicados por seus respectivos fatores excitatórios (pesos sinápticos) e a segunda uma função de

ativação, que, dada as entradas e pesos, determina qual será a saída do neurônio. O axônio seria a saída $y_{(k)}$.

Figura 3 – Representação (simplificada) de um neurônio matemático



Fonte: (Data Science Academy, 2019)

Além dos pesos sinápticos, um *bias* pode ser adicionado à entrada, no somatório da função de combinação, para aumentar o grau de liberdade desta função e, consequentemente, a capacidade de aproximação da rede. O valor de bias é ajustado da mesma forma que os pesos sinápticos e possibilita que um neurônio apresente saída não nula ainda que todas as suas entradas sejam nulas.

Em suma, o neurônio matemático recebe um ou mais sinais de entrada e devolve uma única saída que pode ser usada como saída da rede ou para alimentar outros neurônios da rede em uma camada posterior. Assim, a partir de um conjunto desses neurônios, são formadas as RNAs, que passam por um processo de treinamento, em que aprendem, por exemplo, quais valores de pesos sinápticos devem integrar o modelo.

Um dos primeiros modelos computacionais de RNA foi criado por McCulloch e Pitts (1943) na década de 1940. Esse modelo era fundamentado em matemática e algoritmos. A partir desse modelo duas abordagens de pesquisa surgem: uma voltada aos processos biológicos que ocorrem no cérebro e outra à aplicação de RNAs na IA. De lá para cá, os computadores se tornaram mais e mais potentes, possibilitando cada vez mais a expansão e consolidação desse campo de pesquisa, já que para que as RNAs tenham sucesso, é necessário que se tenha um grande volume de dados a ser passado às mesmas para que elas possam criar modelos

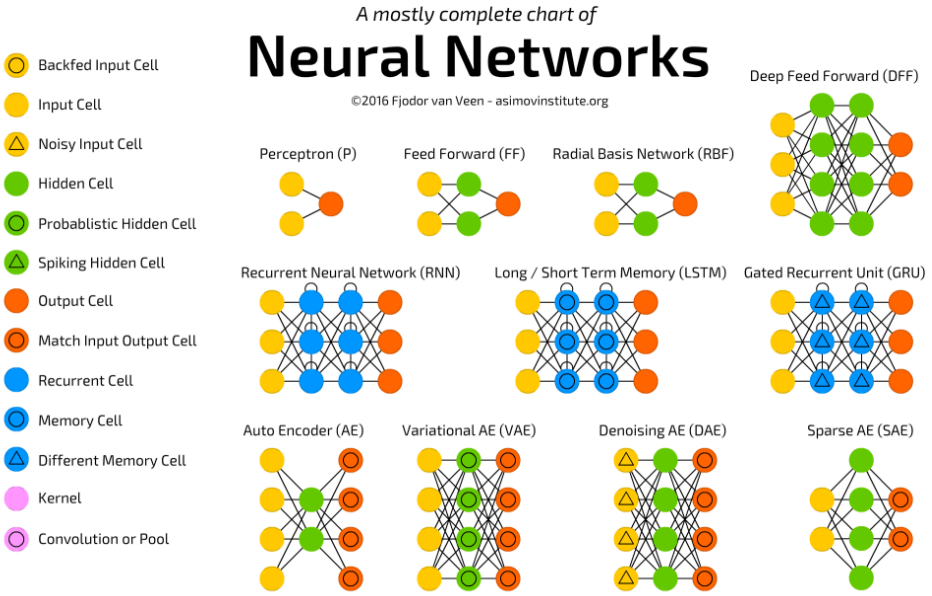
que atinjam altos níveis de precisão. Para fazer o processamento desse grande volume de dados utiliza-se, comumente, GPUs atuando de forma paralela no processamento de operações matemáticas, sobretudo operações envolvendo matrizes e vetores, elementos presentes nos modelos de RNAs.

Uma das ramificações dentro de *machine learning* que indicam essa expansão das RNAs é a área de *deep learning*.

2.1.4 *Deep learning*

Os neurônios que formam uma RNA podem ser conectados de diferentes formas, ou seja, as RNAs podem seguir diferentes topologias (Figura 4).

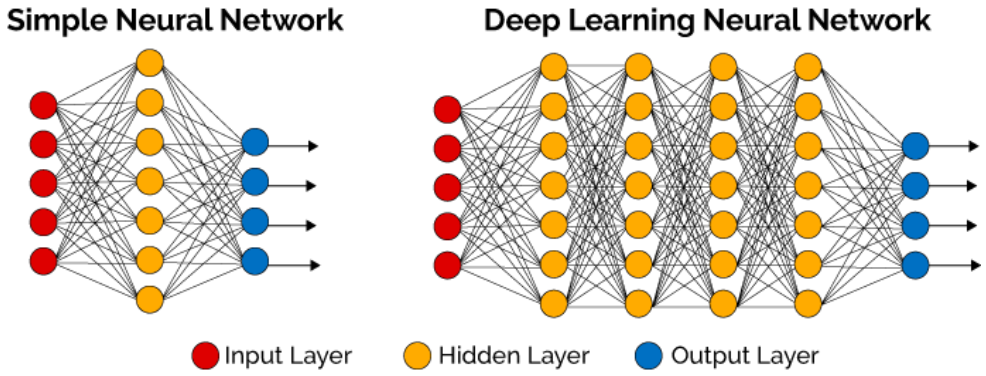
Figura 4 – Topologias de redes neurais artificiais



Fonte: (Data Science Academy, 2019)

Nesse contexto, *deep learning* compreende uma classe de RNAs que usam múltiplas camadas de neurônios para progressivamente extrair alto nível de informações dos dados de entrada brutos. Por terem um número de camadas maior em relação a RNAs tradicionais (Figura 5), elas são chamadas de RNAs profundas, e o processo de aprendizado das mesmas de *deep learning* (aprendizado profundo).

Figura 5 – Comparativo entre uma RNA simples e uma RNA profunda



Fonte: (Data Science Academy, 2019)

As arquiteturas de *deep learning* mais comuns são *deep neural networks*, *deep belief networks*, *recurrent neural networks* e *convolutional neural networks* (CNNs), sendo empregadas em diversos campos da computação como visão computacional, reconhecimento de fala, processamento de linguagem natural e bioinformática. As RNAs profundas têm se destacado em relação às demais RNAs por solucionarem problemas de significativa complexidade como, por exemplo, ganhar de um jogador profissional de Go, aprender a dirigir um carro, diagnosticar câncer de pele e autismo, falsificar arte e detectar objetos em imagens. Especialmente para esse último tipo de tarefa, as CNNs são particularmente excepcionais.

2.1.5 *Convolutional Neural Networks (CNNs)*

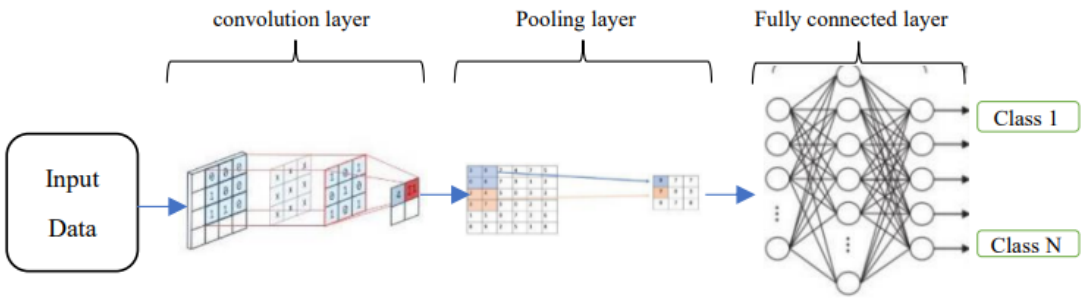
As *convolutional neural networks* (CNNs) foram introduzidas por Yann LeCun no final dos anos 1980 e início dos 1990, com destaque para o modelo LeNet-5, projetado para reconhecimento de dígitos em imagens como, por exemplo, linhas digitáveis em um boletos (LeCun et al., 1998). Desde então, têm evoluído significativamente, ganhando maior capacidade e complexidade, impulsionadas pela crescente disponibilidade de dados e poder computacional, sobretudo das GPUs.

As CNNs são um tipo específico de RNAs amplamente utilizadas no campo de *deep learning*, especialmente para tarefas relacionadas a processamento de imagens. As aplicações das CNNs, incluem:

- **Classificação de imagens**: identificação da classe de uma imagem (e.g., ícone ou botão).
- **Detecção de objetos**: localização e identificação de múltiplos objetos em uma imagem (e.g., elementos de design em uma IU).
- **Segmentação de objetos**: determinação dos pixels de um determinado objeto em uma imagem (e.g., segmentação de imagens médicas).

As CNNs são particularmente eficazes para o processamento de imagens devido a sua capacidade de explorar propriedades espaciais e hierárquicas dos dados visuais.

Figura 6 – Pipeline de uma *Convolutional Neural Network* (CNN)



Fonte: (Shiri et al, 2023)

A figura 6 explicita como esse processamento (Goodfellow et al., 2016) ocorre:

- ***Input Data*:** o processamento inicia com a imagem de entrada, que pode ser representada como uma matriz de valores de pixels. Os valores de pixels carregam informações sobre a intensidade de luz ou cor em cada ponto da imagem.
- ***Convolution Layer***: nesta camada são aplicados filtros à imagem de entrada para extrair características importantes como bordas, texturas ou outros padrões. Os filtros varrem a matriz de pixels, processo ao qual se dá o nome de convolução, e produzem mapas de características que destacam os diferentes aspectos da imagem.
- ***Pooling Layer***: esta camada é usada para reduzir a dimensionalidade dos mapas de características, tornando o processamento mais eficiente e robusto a variações pequenas na posição dos objetos.

- ***Fully Connected Layer***: por fim, os dados são condensados em um vetor e passados para uma ou mais camadas densamente conectadas. Essas camadas funcionam como uma RNA tradicional, combinando as informações extraídas para tomar decisões.

Assim, por meio dessa extração de padrões de imagens, como bordas e formas, usando camadas de convolução e *pooling*, as CNNs são ideais para classificação, segmentação e detecção de objetos em imagens.

2.2 *DESIGN* DE INTERFACE

O *design* de interface visa maximizar a usabilidade e a experiência do usuário, tornando a interação do usuário com o aplicativo eficaz, eficiente e satisfatória (Nielsen, 1993). Nesse contexto, usabilidade é entendida como a extensão em que um produto pode ser usado por usuários específicos para atingir metas especificadas com eficácia, eficiência e satisfação em um contexto específico de uso (ISO 9241-11, 2018). A experiência de usuário corresponde às percepções e respostas de uma pessoa que resultam do uso de um produto, sistema ou serviço (ISO 9241-210, 2006).

No contexto de aplicativos móveis, a usabilidade, sobretudo, é um fator crítico para o sucesso do que é desenvolvido (Yin et al., 2014) (Nascimento et al., 2016). Como os telefones celulares oferecem novas formas de interação (via gesto, sensores, câmera, voz, etc.), seus modelos de interação podem diferir significativamente dos tradicionais (Wasserman, 2010). Com efeito, celulares apresentam requisitos de tamanho e portabilidade com significativas limitações, bem como maneiras estranhas para a entrada de dados. Os telefones *touchscreen*, em especial, representam novos desafios por falta de feedback tátil, tamanho, etc. (Balagtas-Fernandez et al., 2009). Além disso, esses dispositivos são usados por uma ampla gama de pessoas que têm diferentes objetivos, os usam a qualquer momento e em qualquer lugar, de modo que esses fatores fazem com que a complexidade inerente ao design de interface seja maior (Huang, 2009). Isso, concomitantemente, torna a usabilidade um atributo qualitativo ainda mais importante para os aplicativos de *software* de telefones celulares.

2.3 *DESIGN* DE INTERFACE COM APP INVENTOR NA EDUCAÇÃO BÁSICA

O App Inventor é um ambiente de programação baseado em blocos que permite a qualquer pessoa desenvolver aplicativos para dispositivos Android e que pode ser acessado e utilizado on-line de forma gratuita. Sendo usado por uma grande gama de pessoas de todas as idades e origens e tendo mais de 400 mil usuários únicos ativos mensalmente de 195 países diferentes e que criam quase 22 milhões de aplicativos móveis, ele facilita a criação de aplicativos, principalmente para os menos experientes, além de também fornecer feedback imediato ao desenvolvedor por meio da conexão do celular desse com os servidores que o hospedam, permitindo a execução, em tempo real, do aplicativo que está sendo criado. Desse modo, é adotado no ensino de computação para alunos da Educação Básica, que entram em contato com esse campo por meio da programação de aplicativos.

Os aplicativos criados com o App Inventor possuem diversos componentes visuais, como botões, legendas etc e não visuais como câmera, gravador etc.), que são controlados por meio de um conjunto de blocos que sintetizam a lógica dos aplicativos. No Quadro 1 são elencados os componentes visuais que podem ser usados na criação dos aplicativos.

Quadro 1 – Componentes de design de interface do App Inventor

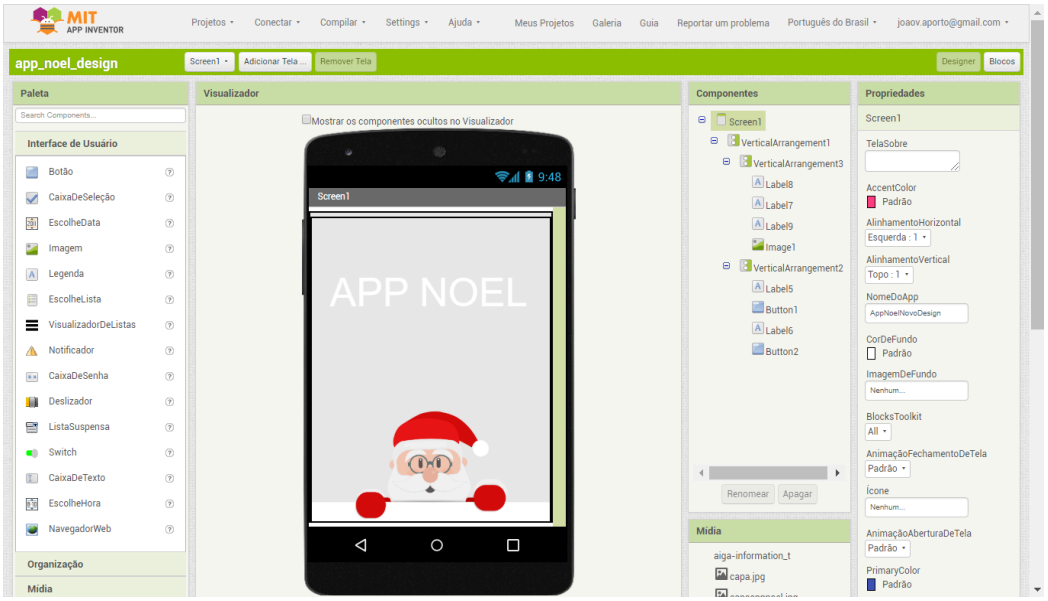
Componente	Descrição
Botão	Componente com a capacidade de detectar cliques. A partir de um clique realiza alguma ação.
Caixa de seleção	Os componentes da caixa de seleção podem detectar toques do usuário e podem alterar seu estado booleano em resposta. Um componente de caixa de seleção gera um evento quando o usuário o toca.
EscolheData	Um botão que, quando clicado, inicia uma caixa de diálogo pop-up para permitir que o usuário selecione uma data.
Imagem	Componente para exibir imagens.
Legenda	Componente usado para mostrar texto.
EscolheLista	Um botão que, quando clicado, exibe uma lista de textos para o usuário escolher.
VisualizadorDeLista	Um componente visível que permite colocar uma lista de elementos na tela para exibir.

Notificador	Componente que exibe caixas de diálogo de alerta, mensagens e alertas temporários
CaixaDeSenha	Os usuários inserem senhas em um componente de caixa de texto de senha, que oculta o texto que foi digitado nele.
Tela	Componente de nível superior contendo todos os outros componentes do programa.
Deslizador	Uma barra de progresso que adiciona um polegar arrastável. O usuário pode tocar no polegar e arrastar para a esquerda ou para a direita para definir a posição do polegar do controle deslizante.
ListaSuspensa	Esse componente exibe um pop-up com uma lista de elementos.
Switch	Gera um evento quando o usuário o clica. Há muitas propriedades que afetam sua aparência que podem ser definidas no Component Designer ou no Editor de blocos.
CaixaDeTexto	Componente em que os usuários inserem texto.
EscolheHora	Um botão que, quando clicado, inicia uma caixa de diálogo pop-up para permitir que o usuário selecione uma hora.

Fonte: elaborado pelo autor

O fluxo padrão de criação de um aplicativo no App Inventor consiste em dois passos. No primeiro, os componentes visuais e não visuais, presentes no menu lateral esquerdo, são selecionados (clicando-se neles ou os arrastando) para uma interface de usuário inicialmente vazia, de modo que esses podem ter suas propriedades (tamanho, cor, fonte, alinhamento etc) alteradas no menu lateral direito. Esse processo de criação das IUs do aplicativo é feito no no Component Designer (Figura 7).

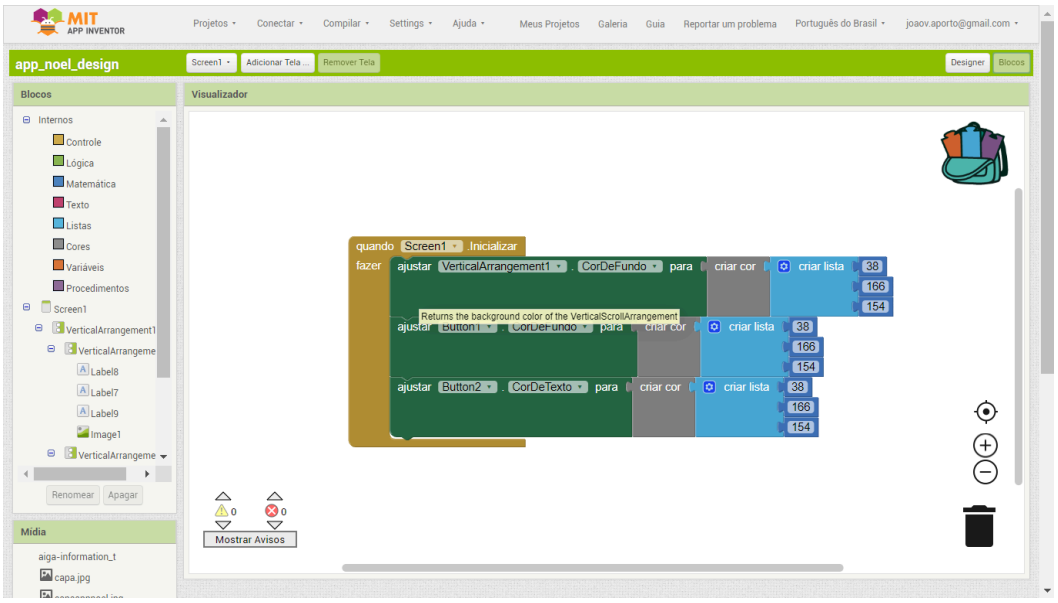
Figura 7 – Component Design do App Inventor



Fonte: elaborado pelo autor

No segundo passo, a lógica do aplicativo é implementada por meio de blocos visuais que representam conceitos de programação como iteração, seleção, procedimentos, funções etc. Blocos específicos para os componentes selecionados no Component Designer que podem ser usados para a configuração de eventos e ações sobre os mesmos como, por exemplo, o clique em botão (Mustafaraj et al., 2017). O funcionamento do aplicativo é definido no Editor de Blocos (Figura 8).

Figura 8 – Editor de Blocos do App Inventor



Fonte: elaborado pelo autor

Os arquivos de código-fonte de aplicativos do App Inventor são salvos automaticamente na nuvem e podem também ser exportados como arquivos *.aia*. Um arquivo *.aia* é um conjunto de arquivos compactados que inclui um arquivo de propriedades do projeto, arquivos de mídia usados pelo aplicativo e, para cada tela do aplicativo, um arquivo *.bky* e um arquivo *.scm*. O arquivo *.bky* contém uma estrutura XML com os blocos de programação usados na lógica do aplicativo, e o arquivo *.scm* contém uma estrutura JSON descrevendo os componentes usados (Mustafaraj et al., 2017).

3. ESTADO DA ARTE

Neste capítulo é apresentado o levantamento do estado da arte realizado por meio de um mapeamento sistemático da literatura existente em relação a modelos existentes que usam *machine learning* para detecção de elementos de IUs, utilizando, para tanto, o método proposto por Petersen (2008).

3.1 DEFINIÇÃO DO PROTOCOLO DE MAPEAMENTO

Para levantar o estado da arte objetivamos responder a seguinte pergunta: “Quais modelos existem para detectar elementos visuais em interfaces de usuário de aplicativos Android usando *Machine Learning*?”. Essa pergunta de pesquisa foi decomposta nas seguintes perguntas de análise:

PA1. Quais elementos são detectados e de qual tipo de *software*?
PA2. Qual(is) conjunto(s) de dados é(são) usado(s) e quais as características desse(s) conjunto(s)?
PA3. Qual(is) algoritmo(s), modelo(s), e framework(s) de *Machine Learning* são usados?
PA4. Quais são os resultados obtidos e como é medida a qualidade desses resultados?

Critérios de inclusão/exclusão. Foram examinados artigos, escritos em inglês e português, publicados em periódicos ou anais de conferências. Consideramos artigos publicados de 2009 a 2019. O ano de início escolhido foi 2009 por ser o ano em que a ImageNet (Deng et al., 2009) foi lançada, marcando uma nova fase do campo de *machine learning* em que o hardware e as técnicas voltados a esse campo passaram a trazer resultados significativos para resolução de problemas que não eram facilmente solucionados por técnicas convencionais. Consideramos também apenas artigos que abordaram a detecção dos elementos visuais por meio de *machine learning*. Por outro lado, excluímos:

- Artigos cujo conteúdo não fornecia informações suficientes para responder uma ou mais perguntas de pesquisa;
- Artigos que não continham uma proposta de detecção de elementos de IUs com técnicas de *machine learning*.

Critério de qualidade. A qualidade dos estudos presentes nas publicações foi avaliada superficialmente, considerando-se somente aqueles que apresentaram informações substanciais alinhadas à pergunta de pesquisa e às perguntas de análise.

Fontes de dados e estratégias de pesquisa. Analisamos artigos presentes na web por meio das principais bases de dados e bibliotecas digitais do campo da computação (ACM Digital Library, IEEE Xplore Digital Library, Scopus, SpringerLink e Wiley) com livre acesso por meio do Portal CAPES. Além dessas bases, com o intuito de cobrir uma gama maior de publicações, também foram conduzidas pesquisas no Google Scholar e Google, que indexam um grande conjunto de dados de diversas fontes de produção científica distintas (Haddaway et al., 2015).

Para que a *string* de busca pudesse levar a uma resposta satisfatória da pergunta de pesquisa, criou-se uma *string* com base nos termos da pergunta e foi executado um processo de calibração dessa *string* que consistiu em algumas buscas preliminares. Essas buscas preliminares, retornaram um número muito pequeno de artigos. Assim, durante a calibração da *string*, priorizamos termos que a tornassem o mais abrangente possível. Para tanto, definimos sinônimos e termos relacionados aos termos de busca que compõem a *string* e não restringimos a busca somente a artigos referenciando aplicativos Android, conforme indicado no Quadro 2.

Quadro 2 – Termos de busca, seus sinônimos e termos relacionados

Termo de busca	Sinônimos e palavras relacionadas
detect	recognize, detection, extract
elements	components, widget, text, image, icon, button
user interfaces	user interface, UI, UIs, GUI
apps	app, application, android, ios
machine learning	deep learning, neural network, ANN, artificial intelligence, AI, CNN, computer vision

Fonte: elaborado pelo autor

Após a calibração, definiu-se uma *string* de busca específica para cada uma das bases de dados consideradas. A relação dessas *strings* com as respectivas bases é apresentada no Quadro 3.

Quadro 3 – *Strings* de buscas usadas nas bases de dados

Base de dados	*String* de busca
ACM Digital Library	(detect* OR recogni* OR extract*) AND (element OR component OR widget OR text OR image OR icon OR button) AND ("user interface" OR "user interfaces" OR ui OR gui) AND (app OR android OR ios) AND ("machine learning" OR "deep learning" OR "neural network" OR "neural networks" OR "artificial intelligence" OR cnn OR "computer vision")
Google	(detect OR recognize OR extract) AND (element OR component OR widget OR text OR image OR icon OR button) AND ("user interface" OR "user interfaces" OR ui OR gui) AND (app OR android OR ios) AND ("machine learning" OR "deep learning" OR "neural network" OR "neural networks" OR "artificial intelligence" OR cnn OR "computer vision")
Google Scholar	(detect* OR recogni* OR extract*) AND (element OR component OR widget OR text OR image OR icon OR button) AND ("user interface" OR "user interfaces" OR ui OR gui) AND (app OR android OR ios) AND ("machine learning" OR "deep learning" OR "neural network") (detect* OR recogni* OR extract*) AND (element OR component OR widget OR text OR image OR icon OR button) AND ("user interface" OR "user interfaces" OR ui OR gui) AND (app OR android OR ios) AND ("neural networks" OR "artificial intelligence" OR cnn) (detect* OR recogni* OR extract*) AND (element OR component OR widget OR text OR image OR icon OR button) AND ("user interface" OR "user interfaces" OR ui OR gui) AND (app OR android OR ios) AND ("computer vision")

IEEE Xplore Digital Library	(detect* OR recogni* OR extract*) AND (element OR component OR widget OR text OR image OR icon OR button) AND ("user interface" OR "user interfaces" OR ui OR gui) AND (app OR android OR ios) AND ("machine learning" OR "deep learning" OR "neural network" OR "neural networks" OR "artificial intelligence" OR cnn OR "computer vision")
Scopus	TITLE-ABS-KEY ((detect* OR recogni* OR extract*) AND (element OR component OR widget OR text OR image OR icon OR button) AND ("user interface" OR "user interfaces" OR ui OR gui) AND (app OR android OR ios) AND ("machine learning" OR "deep learning" OR "neural network" OR "neural networks" OR "artificial intelligence" OR cnn OR "computer vision")) AND (LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "ar") OR LIMIT-TO (DOCTYPE , "re") OR LIMIT-TO (DOCTYPE , "ch") OR LIMIT-TO (DOCTYPE , "cr") OR LIMIT-TO (DOCTYPE , "sh")) AND (LIMIT-TO (PUBYEAR , 2019) OR LIMIT-TO (PUBYEAR , 2018) OR LIMIT-TO (PUBYEAR , 2017) OR LIMIT-TO (PUBYEAR , 2016) OR LIMIT-TO (PUBYEAR , 2015) OR LIMIT-TO (PUBYEAR , 2014) OR LIMIT-TO (PUBYEAR , 2013) OR LIMIT-TO (PUBYEAR , 2012) OR LIMIT-TO (PUBYEAR , 2011) OR LIMIT-TO (PUBYEAR , 2010) OR LIMIT-TO (PUBYEAR , 2009)) AND (LIMIT-TO (LANGUAGE , "English") OR LIMIT-TO (LANGUAGE , "Portuguese"))
SpringerLink	(detect* OR recogni* OR extract*) AND (element OR component OR widget OR text OR image OR icon OR button) AND ("user interface" OR "user interfaces" OR ui OR gui) AND (app OR android OR ios) AND ("machine learning" OR "deep learning" OR "neural network" OR "neural networks" OR "artificial intelligence" OR cnn OR "computer vision")

Wiley Online Library	(detect* OR recogni* OR extract*) AND (element OR component OR widget OR text OR image OR icon OR button) AND ("user interface" OR "user interfaces" OR ui OR gui) AND (app OR android OR ios) AND ("machine learning" OR "deep learning" OR "neural network" OR "neural networks" OR "artificial intelligence" OR cnn OR "computer vision")

Fonte: elaborado pelo autor

Nota-se que foram definidas 3 *strings* para a busca no Google Scholar. Essa fragmentação foi feita devido à limitação do campo de busca dessa base.

3.2 EXECUÇÃO DA BUSCA E SELEÇÃO DE RESULTADOS

A busca foi executada durante os meses de setembro e outubro de 2019 pelo autor (Tabela 1). Após a execução da busca em cada uma das bases de dados, foi feita a análise dos resultados com base no título, resumo e palavras-chave. Nos casos de grande quantidade de resultados retornados, foram analisados somente os 100 resultados mais relevantes listados em cada base de dados. Como a busca no Google Scholar foi fragmentada, foram selecionados os primeiros 34 resultados da primeira *string* e os 33 primeiros resultados das duas outras.

A partir da análise do título, resumo e palavras-chave foi possível identificar os resultados potencialmente relevantes e, restringindo-se a esses, executar uma análise do conteúdo completo dos artigos para selecionar os mais relevantes.

Tabela 1 – Número de resultados no processo de seleção de 2019

Base de dados	Número de resultados da busca	Número de resultados analisados com base no título, resumo e palavras-chave	Número de artigos potencialmente relevantes	Número de artigos relevantes
ACM Digital Library	18	18	5	4
Google	289	100	2	2
Google Scholar	51 500	100	11	5
IEEE Xplore Digital Library	34	34	8	4
SpringerLink	2 752	100	1	0

Scopus	47	47	10	5
Wiley Online Library	118	100	0	0
Total	54758	499	37	20
Total sem duplicatas	-	448	19	5

Fonte: elaborado pelo autor

A maior parte dos artigos foi excluída por não abordar especificamente a identificação de elementos de IU de aplicativos móveis, seja como objetivo principal ou secundário. Outros artigos, mesmo abordando a identificação de elementos, foram excluídos por apresentarem informações insuficientes para responder as perguntas de análise. Ao final, 5 artigos foram considerados relevantes (Quadro 4).

Quadro 4 – Artigos relevantes selecionados em 2019

Referência	Artigo
(Moran et al., 2018)	Moran K., Bernal-Cárdenas C., Curcio M., Bonett R. e Poshyvanyk D. “Machine learning-based prototyping of graphical user interfaces for mobile apps”. IEEE Transactions on Software Engineering, 2018.
(Chen et al., 2019)	Chen S., Fan L., Su T., Ma L., Liu Y. e Xu L.. “Automated cross-platform GUI code generation for mobile apps”. In Proc. of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, 2019.
(Chen et al., 2018)	Chen C., Su T., Meng G., Xing Z. e Liu Y. “From UI design image to GUI skeleton: a neural machine translator to bootstrap mobile GUI implementation”. In Proc. of the 40th International Conference on Software Engineering, 2018, 665-676.
(Liu et al., 2018)	Liu T. F., Craft M., Situ J., Yumer E., Mech R. e Kumar R. “Learning Design Semantics for Mobile Apps”. In Proc. of the 31st Annual ACM Symposium on User Interface Software and Technology, Berlin, Germany, 2018, 569-579.
(Xiao et al., 2019)	Xiao X., Wang X., Cao Z., Wang H. e Gao P.. “IconIntent: automatic identification of sensitive UI widgets based on icon classification for Android apps”. In Proc.of the 41st Int. Conf.on Software Engineering, 2019, USA, 257-268.

Fonte: elaborado pelo autor

3.3 REEXECUÇÃO DA BUSCA E SELEÇÃO DE RESULTADOS (2020-2024)

Dada a retomada deste trabalho após um período de tempo extenso, foram executadas novas buscas em setembro de 2024, seguindo o mesmo método proposto por Petersen (2008) e nas mesmas bases, exceto pela base Scopus, que da qual não foi possível obter busca aberta de documentos (Tabela 2).

Tabela 2 – Número de resultados no processo de seleção de 2024

Base de dados	Número de resultados da busca	Número de resultados analisados com base no título, resumo e palavras-chave	Número de artigos potencialmente relevantes	Número de artigos relevantes
ACM Digital Library	4 882	100	8	1
Google	13 400 000	100	3	0
Google Scholar	51 400	100	12	0
IEEE Xplore Digital Library	235	100	5	0
SpringerLink	6 464	100	7	0
Wiley Online Library	2 039	100	6	0
Total	13 465 020	600	41	0
Total sem duplicatas	-	598	40	1

Fonte: elaborado pelo autor

Novamente, grande parte dos artigos foi excluída por não abordar especificamente a identificação de elementos de IU de aplicativos móveis. Ao final, 1 artigo foi considerado relevante (Quadro 5).

Quadro 5 – Artigos relevantes selecionados em 2024

Referência	Artigo
(Degaki et al., 2022)	Degaki R. H., Colonna J. G., Lopez Y., Carvalho J. R. e Silva E. “Real Time Detection of Mobile Graphical User Interface Elements Using Convolutional Neural Networks”. In Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia '22). Association for Computing Machinery, New York, NY, USA, 159–167. https://doi.org/10.1145/3539637.3558044

Fonte: elaborado pelo autor

3.4 EXTRAÇÃO E ANÁLISE DE DADOS

De acordo com as perguntas de análise foram extraídas sistematicamente as informações dos artigos relevantes.

Muitos desses artigos não tinham a detecção dos elementos das IUs como principal fim, mas como um meio, principalmente para a geração de código de GUIs a partir das IUs. Considerando a pergunta de pesquisa que conduziu este trabalho, somente os dados relacionados à detecção de objetos foram extraídos e analisados.

As informações extraídas são apresentadas nos subcapítulos a seguir.

3.4.1 Quais elementos são detectados e de qual tipo de *software*?

À exceção de um artigo (Chen et al., 2019), todos os demais suportam apenas a plataforma Android (Quadro 6). Por serem mais voltados à plataforma Android, a maior parte dos elementos identificados correspondem aos componentes presentes nas hierarquias de elementos dos aplicativos construídos para essa plataforma.

Quadro 6 – Dados extraídos de acordo com pergunta de análise 1

Referência	Elementos detectados	Plataforma
(Moran et al., 2018)	SwitchCompat, ImageSwitcher, TextSwitcher, compoundButton, NumberPicker, ViewSwitcher ToggleButton, RatingBar, SeekBar, Switch, ProgressBar, ImageView, ImageButton, Button, EditText, CheckBox, CheckedTextView	Android
(Chen et al., 2019)	EditText, Button,TextView, entre outros não detalhados	Android e iOS
(Chen et al., 2018)	Widgets e layouts nativos da plataforma Android e widgets de bibliotecas third-party	Android
(Liu et al., 2018)	Advertisement, background image, bottom navigation, button bar, card, checkbox, date picker, drawer, icon, image, input, list item, map view, modal, multi-tab, number stepper, on/off switch, pager indicator, radio button, slider, text, text button, toolbar, video e web view	Android
(Xiao et al., 2019)	Widgets, ícones	Android

(Degaki et al., 2022)	Advertisement, background image, bottom navigation, button bar, card, checkbox, date picker, drawer, icon, image, input, list item, map view, modal, multi-tab, number stepper, on/off switch, pager indicator, radio button, slider, text, text button, toolbar, video e web view	Android

Fonte: elaborado pelo autor

Essa cobertura limitada indica a necessidade de expansão dos modelos para outras plataformas de dispositivos móveis também largamente usadas, como iOS.

3.4.2 Quais conjunto(s) de dados são usados(s) e quais as características desse(s) conjunto(s)?

À exceção de um trabalho (Liu et al., 2018), todos apresentaram conjuntos de dados com um número muito significativo de elementos, com destaque para o trabalho (Chen et al., 2019) com 1 842 580 de *screenshots* de UIs únicas (Quadro 7).

Quadro 7 – Dados extraídos de acordo com pergunta de análise 2 (parte 1)

Referência	Descrição do(s) conjunto(s) de dados	Coleta de dados	Rotulagem
(Moran et al., 2018)	14 382 *screenshots* de IUs únicas e 191 300 componentes de UI rotulados	O conjunto de dados foi coletado de uma maneira totalmente automatizada pela mineração e execução automática dos 250 principais aplicativos Android em cada categoria do Google Play, excluindo categorias de jogos	Foi utilizada uma metodologia que automatiza a rotulagem: repositórios de aplicativos foram minerados para encontrar os executáveis. Posteriormente, foi utilizado um gerador de entradas para GUIs para executar automaticamente os aplicativos minerados. Enquanto os aplicativos eram executados, capturas

			de tela e metadados relacionados às GUIs foram extraídas automaticamente para cada tela observada. Esses metadados foram usados para rotular automaticamente as imagens dos componentes.
(Chen et al., 2019)	O conjunto de dados usado contém 1 842 580 *screenshots* de IUs únicas	Foram minerados 37 251 aplicativos Android e 8 951 aplicativos iOS do Google Play Store e iTunes App Store, respectivamente. Cada aplicativo foi executado em emuladores para a captura dos *screenshots* das IUs	Foram usadas ferramentas de teste de IUs para capturar as informações de tipo dos componentes
(Chen et al., 2018)	185 277 pares de *screenshots* de IUs e GUI *skeletons* (composição hierárquica dos componentes da GUI). 93% dos pares foram usados para treino e os 7% restantes para testes	Foram minerados 6000 aplicativos Android com os maiores números de instalações do Google Play. 5043 aplicativos executaram com sucesso em uma ferramenta desenvolvida pelos autores que automatiza o processo de extração dos *screenshots* e *skeletons* das IUs	Os componentes das IUs presentes nas *screenshots* foram associados de forma automatizada aos componentes nos GUI *skeletons* pela ferramenta desenvolvida pelos autores, chamada Stoat (Su et al, 2017)
(Liu et al., 2018)	Foi utilizado como principal conjunto de dados o RICO, que é um conjunto de dados de aplicativos móveis voltado ao desenvolvimento	O RICO inclui hierarquias de elementos de IUs, o que foi aproveitado no trabalho deste artigo. Para treinar a rede CNN usada	Com o conjunto de potenciais ícones, foi feita a classificação e rotulagem desses ícones utilizando

	orientado a dados de aplicativos móveis, possuindo 72219 IUs de 9772 aplicativos. Para cada IU, são disponibilizados seu *screenshot* (PNG), sua a estrutura hierárquica dos elementos (JSON), um conjunto de interações do usuário com a mesma (JSON), um conjunto de animações capturando efeitos de transição em resposta às interações do usuário (GIF) e uma representação vetorial aprendida de seu layout (JSON e NPY).	na classificação de ícones, os pesquisadores extraíram imagens das IUs do RICO de acordo com uma heurística para definir potenciais ícones. A heurística consistiu em pegar somente componentes de imagem que tivessem as propriedades clickable e visible-to-user verdadeiras, ocupassem menos do que 5% da área total da tela e cuja proporção de tela fosse maior que 0,75. Por fim, foram eliminados componentes duplicados e a quantidade de potenciais ícones atingida foi de 73499.	métodos automáticos (iterative open coding) e manuais (pesquisadores associaram ícones a classes de ícones).
(Xiao et al., 2019)	O conjunto de dados usado foi feito pelos próprios autores e provém de duas fontes, Google Imagens e Google Play Store, sendo divido em dois, conjunto de treino e conjunto de teste, o primeiro com 1 576 ícones e o segundo com e 5 791 ícones	Para o conjunto de dados de treino foram coletadas imagens do Google Imagens e de aplicativos da Google Play Store. Para o conjunto de dados de teste foram coletados os aplicativos mais populares da Google Play Store (diferentes dos usados no conjunto de testes).	A rotulagem foi feita de forma manual para ambos os conjuntos de dados
(Degaki et al., 2022)	Foi utilizado como conjunto de dados base o RICO, derivando-se dele 3 conjuntos de dados, um de elementos clicáveis e não	Utilizando-se de um script próprio para adequação do RICO ao formato COCO, os autores	Automática por meio de script próprio

	clicáveis, outro de componentes de IU e um último de ícones	chegaram nos 3 conjuntos de dados	

Fonte: elaborado pelo autor

Nota-se que para formar esses grandes conjuntos de dados foram usados processos inteiramente automatizados ou parcialmente. Nesses processos, ferramentas são usadas para explorar e minerar aplicativos em repositórios de aplicativos e para executá-los, simulando o comportamento de um usuário real para a captura de informações de interação e dos *screenshots* das IUs. Quanto à execução da rotulagem, em alguns casos foram usadas ferramentas (Moran et al., 2018) (Chen et al., 2019) (Chen et al., 2018) (Degaki et al., 2022) e em outros (Liu et al., 2018) (Xiao et al., 2019) a execução foi feita de forma manual pelos pesquisadores.

Todos os conjuntos utilizaram como fonte de coleta de aplicativos a Google Play Store, que reúne milhares de aplicativos Android. Um fator problemático que pode decorrer dessa cobertura limitada é o *overfitting*, tendo em vista que somente IUs de um determinado tipo poderão ter seus elementos satisfatoriamente reconhecidos. Com aplicativos criados com App Inventor, por exemplo, pode ser que esses modelos não apresentem bons resultados.

Quadro 8 – Dados extraídos de acordo com pergunta de análise 2 (parte 2)

Referência	Pré-processamento	Disponibilidade	Endereços(s) para acessar conjunto
(Moran et al., 2018)	*Data Augmentation*, *Data Segmentation*	Disponível para download	ReDraw dataset
(Chen et al., 2019)	-	Conjunto de dados não disponibilizado	-
(Chen et al., 2018)	-	Disponível para download	-
(Liu et al., 2018)	As técnicas usadas para pré-processamento foram: conversão para escala de	Disponível para download até ao menos 2022	RICO dataset

	cinza, centralização *feature-wise* e desvio padrão, translação e ZCA		
(Xiao et al., 2019)	As técnicas usadas para pré-processamento foram: redimensionamento de imagem, conversão para escala de cinza, inversão de cores, ajuste de contraste e conversão de opacidade	Conjunto de dados não disponibilizado	-
(Degaki et al., 2022)	Filtro de instâncias inconsistentes e filtro de anotações ruidosas	Disponível em repositório público	Repositório

Fonte: elaborado pelo autor

Somente três trabalhos, (Moran et al., 2018), (Liu et al., 2018) e (Degaki et al., 2022), disponibilizaram os conjunto de dados de treinamento para download, os dois primeiros contendo *screenshots* das IUs de aplicativos Android. elementos rotulados das IUs e dados dos aplicativos, propiciando o reuso dos mesmos para a criação de novos modelos e o último um conjunto de dados no formato COCO. O ReDraw se distingue do RICO por conter os elementos das IUs recortados e rotulados, prontos a serem usados como entrada para algoritmos de *machine learning*. O RICO, por outro lado, contém os *screenshots* das IUs e as estruturas hierárquicas dessas IUs contendo anotações semânticas sobre os elementos.

3.4.3 Qual(is) algoritmo(s), modelo(s), e framework(s) de *Machine Learning* são usados?

A maioria dos modelos encontrados usa algoritmos de aprendizado supervisionado para a detecção dos elementos das IUs (Moran et al., 2018) (Liu et al., 2018) (Xiao et al., 2019) baseados em uma *convolutional neural network* (CNN) (Quadro 9). Entretanto, somente um dos trabalhos (Chen et al., 2019) apresentou detalhes arquiteturais sobre a rede. Da mesma forma, somente um trabalho apresentou de forma detalhada quais foram os *frameworks* usados para a criação e treinamento da rede (Moran et al., 2018).

Quadro 9 – Dados extraídos de acordo com pergunta de análise 3

Referência	Algoritmo(s)	Modelo(s)	Framework(s)
(Moran et al., 2018)	Foi usado treinamento supervisionado com rede neural	Foi usada uma CNN com arquitetura semelhante a da AlexNet, com 3 camadas convolucionais.	A CNN foi implementada no MATLAB usando os toolkits Neural Network, Parallel Computing e Computer Vision.
(Chen et al., 2019)	Foi usado treinamento supervisionado com rede neural	Foi usada uma CNN cuja a especificação não foi detalhada	-
(Chen et al., 2018)	Foi usado treinamento supervisionado com rede neural	Foram usadas uma CNN e uma RNN. A arquitetura de CNN usada contém como as principais camadas uma camada convolucional (Conv) e uma camada de *pooling* (Pool), seguindo o padrão Conv-Pool. Foram empilhadas 6 camadas Conv-Pool para criar uma CNN profunda. Na primeira camada Conv foram usados 64 neurônios e em cada camada subsequente foi usado o dobro de neurônios da anterior. Na camada Pool foram usadas unidades de *pooling* de tamanho 2 × 2 aplicadas com um *stride* de 2. Não foram usadas camadas totalmente conectadas com o objetivo de preservar a localidade dos recursos	O modelo foi implementado com base no *framework* Torch

		da CNN para codificar suas informações de layout espacial posteriormente. A RNN usada foi uma Long Short-Term Memory (LSTM)		
(Liu et al., 2018)	Foi usado treinamento supervisionado com rede neural	Foi usada uma adaptação da arquitetura de CNN que tinha os melhores resultados para o conjunto de dados CIFAR-100 à época para a classificação dos ícones. A rede usada possuía 17 camadas convolucionais. As duas últimas camadas foram inteiramente conectadas, uma com 512 neurônios e uma camada final de ativação de softmax com 99 neurônios. Além disso, para diferenciar imagens diferentes de ícones, foi treinado um detector de anomalia para sinalizar ativações anômalas na última camada.	Scikit-Learn	
(Xiao et al., 11)	SIFT (scale-invariant feature transform)	Foi criada uma adaptação do algoritmo de reconhecimento de objetos em imagens SIFT		-
(Degaki et al., 2022)	Foi usado treinamento supervisionado com rede neural	Foi utilizado o modelo de CNN YOLO em sua quinta versão.		-

Fonte: elaborado pelo autor

A prevalência do uso de CNNs nesses modelos evidencia a adequação desse tipo de RNA aos problemas de predição envolvendo imagens como elementos do conjunto de dados para treinamento.

3.4.4 Quais são os resultados obtidos e como é medida a qualidade desses resultados?

Todos os modelos apresentaram bons resultados (Quadro 10), podendo-se observar que aqueles que foram comparados com técnicas alternativas de detecção dos elementos (SVM, Regression, Logistic etc) as superaram. No entanto é possível que a precisão dos modelos (a principal medida utilizada), ainda possa ser elevada, já que em nenhum dos casos chegou a atingir 95%.

Quadro 10 – Dados extraídos de acordo com pergunta de análise 4

Referência	Medidas de desempenho	Resultados das medidas de desempenho	Resultados e conclusões gerais sobre o modelo
(Moran et al., 2018)	Foi utilizada a precisão média *top-1* da classificação feita pela CNN	A precisão top-1 total da CNN (com base em números brutos de componentes classificados) foi de 91,1%	A precisão de 91,1% foi comparada com a precisão da técnica alternativa *Support Vector Machine* (SVM), verificando-se que a precisão da CNN superou a precisão da técnica alternativa
(Chen et al., 2019)	*Accuracy*, *precision* e *recall*	Foi atingido mais do que 85% de *accuracy*	A *accuracy* de 85% foi comparada com a de outras técnicas alternativas (SVM, Regression, Logistic e K-nearest neighbors), e superou a de todas elas, que atingiram entre 20% e 70% de *accuracy*
(Chen et al., 2018)	Foi medida a porcentagem de	A correspondência exata entre UIs e	Apesar da precisão baixa, o modelo tem

	correspondência exata entre os *skeletons* das IUs e *skeletons* gerados pelo modelo	*skeletons* foi de 60,28%.	potencialidade para ser aplicado na conversão de IUs em *skeletons*.
(Liu et al., 2018)	Foram usadas como medidas de desempenho para a classificação de ícones a precisão e o *recall* da CNN treinada. A precisão geral no conjunto de dados de teste diminui de 94% para 91% e o recall diminui de 86% para 79% com a detecção de anomalias. Contudo, a precisão da CNN aumenta de 87% a 91%.	Para demonstrar a utilidade da abordagem em escala, Foram feitas anotações semânticas para as 72 mil IUs do conjunto de dados RICO, alcançando-se a anotação para 78% dos elementos existentes	A arquitetura de CNN usada não tem bom desempenho em ícones com pouca presença no conjunto de treinamento. Como resultado, não foi possível treinar um classificador multiclasse que reconhecesse todos os tipos de ícones que identificados originalmente
(Xiao et al., 11)	Foi utilizada como medida de desempenho o F-score médio. Para a classificação de ícones o F-score médio foi de 87,7% e para a classificação de ícones de texto foi de 89,8%	Avaliações do IconIntent em 150 aplicativos do Google Play mostraram que ele pode detectar 248 widgets de IUs de 97 aplicativos, alcançando uma precisão de 82,4%. Quando combinado ao SUPOR (identificador de widget de IUs do baseado na análise de texto), foi possível detectar 487 widgets (melhoria de 101,2% em relação ao SUPOR somente)	Autores declaram serem os primeiros a investigar a intenção dos ícones de IUs de aplicativos móveis e estudar o uso deles nos widgets, propondo uma nova estrutura, IconIntent, para associar ícones a correspondentes widgets de IUs e classificar as intenções dos ícones em categorias predefinidas

(Degaki et al., 2022)	Foram utilizadas como medidas de desempenho precisão, *recall* e mAP.	Os autores relatam que todos os resultados apresentam valores satisfatórios de mAP@.5, no entanto, esse valores de precisão e recall ficam abaixo de 0.7, indicando potencial para melhoria	Autores declaram que independentemente dos resultados do modelo o principal objetivo do trabalho foi a criação dos conjuntos de dados

Fonte: elaborado pelo autor

3.5 DISCUSSÃO

Ao final deste mapeamento sistemático da literatura podemos concluir que o campo de detecção de elementos de IUs de aplicativos móveis ainda era incipiente nos meados de 2019 e teve alguns avanços até 2024, mas ainda com poucos poucos trabalhos voltados a esse fim. Os trabalhos encontrados apresentam modelos capazes de detectar elementos de IUs com boa precisão por meio de *Machine Learning* em relação a outras técnicas e abordagens encontradas na literatura. Entretanto, ainda é oportuna a criação de novos modelos que atinjam melhores resultados nas medidas de desempenho.

3.5.1 Ameaças à validade

Em uma revisão ou mapeamento sistemático da literatura, existem algumas ameaças à validade dos resultados. Assim, identificamos potenciais ameaças e aplicamos estratégias para atenuá-las quando possível, minimizando seus impactos.

Viés de publicação. Uma das ameaças mais comuns às revisões sistemáticas é a tendência de artigos com resultados positivos serem mais publicados do que artigos com resultados negativos (Kitchenham et al., 2007). Apesar disso, não consideramos essa uma séria ameaça, já que não foram encontrados muitos resultados, sejam positivos ou negativos, diretamente relacionados à pergunta de pesquisa.

Identificação de estudos. Outra ameaça a ser considerada é a omissão de estudos relevantes. Para evitá-la, buscamos construir uma *string* de busca mais abrangente possível, incrementando-a com sinônimos encontrados em artigos relacionados à questão de pesquisa. Realizamos também as buscas em diversas

bases (as principais no campo da computação), no Google e Google Scholar, ampliando o número de publicações cobertas nos resultados.

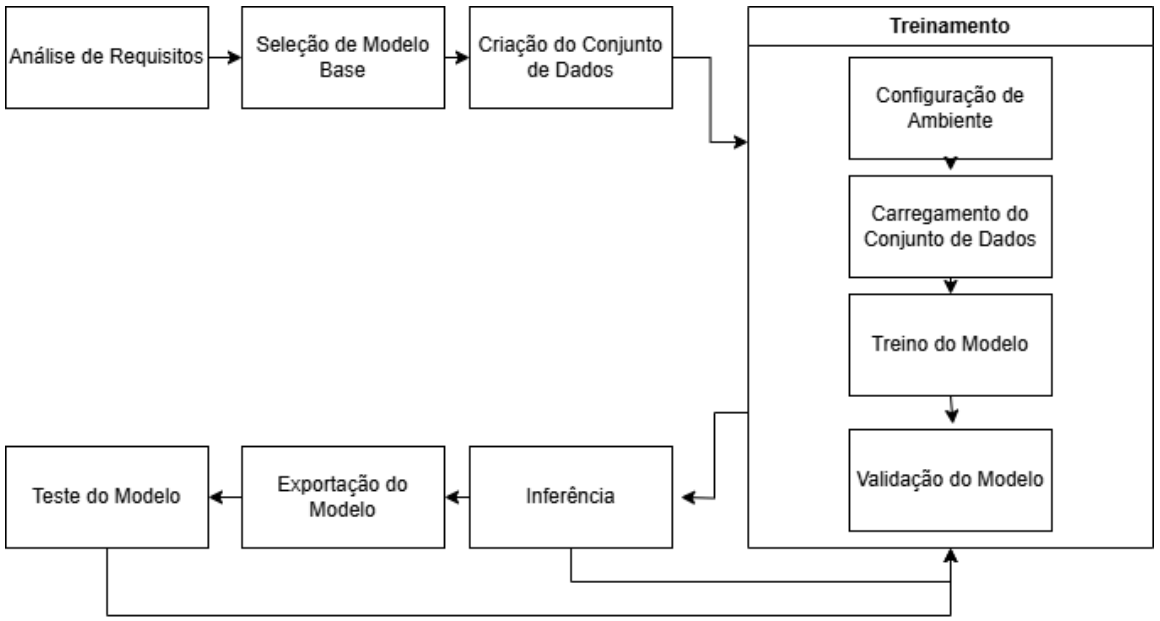
Seleção de estudos e extração de dados. Uma seleção de estudos incorreta é uma das ameaças existentes nesse tipo de pesquisa, assim como falhas na extração de dados. Desse modo, empregamos rigor tanto na definição dos critérios de inclusão/exclusão quanto no processo de seleção das publicações, que foi realizado pelo autor e supervisionado pela orientadora, discutindo-se até um consenso, quando necessário, o que se adequava ou não à questão de pesquisa.

4. DESENVOLVIMENTO DO MODELO “TifSX” PARA EXTRAÇÃO SEMÂNTICA

Nesta seção é apresentado o modelo desenvolvido neste trabalho, nomeado “TifSX”, que é um acrônimo recursivo para “TifSX is for Semantics eXtraction”.

O processo de desenvolvimento do modelo se deu conforme as etapas expressas na Figura 9, que são detalhadas a seguir.

Figura 9 – Fluxo de desenvolvimento do modelo TifSX



Fonte: elaborado pelo autor

A etapa de “Análise de Requisitos” é apresentada na subseção 4.1; a etapa de “Seleção de Modelo Base” na subseção 4.2; a etapa de “Criação do Conjunto de Dados” na subseção 4.3; a etapa de “Treinamento” é abordada na subseção 4.4; a etapa de “Inferência” na subseção 4.5; a etapa de “Exportação do Modelo” na subseção 4.6; por fim, a etapa de “Teste do Modelo” é apresentada na seção 5.

4.1 ANÁLISE DE REQUISITOS

Como foi definido na seção 1.2, o objetivo deste trabalho é criar um modelo capaz de identificar elementos de uma IU e extrair a semântica dos mesmos, para que futuramente o modelo seja integrado à ferramenta CodeMaster, viabilizando a avaliação das IUs de um aplicativo. Assim, foram analisados os requisitos para a criação desse modelo de extração.

Os requisitos funcionais para o modelo são dados no Quadro 11 e os não-funcionais no Quadro 12.

Quadro 11 – Requisitos funcionais do modelo de extração de semântica de elementos de IU

ID	Requisito	Descrição	Artefato de entrada	Artefato de saída
RF01	Identificar elementos de IU	O modelo deve ser capaz de identificar elementos de design pertencentes a uma IU	*Screenshot* da IU	Elementos presentes na IU
RF02	Extrair significado de elementos de IU	O modelo deve ser capaz de reconhecer o significado de cada um dos elementos pertencentes a uma IU, o que inclui reconhecer qual o tipo de elemento e qual sua funcionalidade na IU, a depender do contexto e posição em que ele aparece	*Screenshot* da IU	Elementos presentes na IU e seus significados
RF03	Identificar posição dos elementos	O modelo deve ser capaz de identificar onde cada um dos elementos reconhecidos de uma IU se encontra	*Screenshot* da IU	Posições dos elementos presentes na IU

Fonte: elaborado pelo autor

Quadro 12 – Requisitos não-funcionais do modelo para extração de semântica de elementos de IU

ID	Requisito	Descrição
RNF01	Extração da semântica de IUs do App Inventor	O modelo deve ser capaz de extrair a semântica das IUs dos aplicativos criados com App Inventor
RNF02	Integração com CodeMaster	O modelo deve ser capaz de ser integrado à ferramenta CodeMaster

RNF03	Tempo para extração	O modelo não pode demorar mais que 100 ms para extrair a semântica dos elementos de uma IU

Fonte: elaborado pelo autor

4.2 MODELO PARA EXTRAÇÃO DA SEMÂNTICA DE *DESIGN* DE ELEMENTOS DE IU

Levando em consideração os 3 requisitos funcionais necessários ao modelo, vê-se a necessidade de um modelo capaz de identificar objetos e simultaneamente localizar suas posições em imagens em um tempo relativamente baixo.

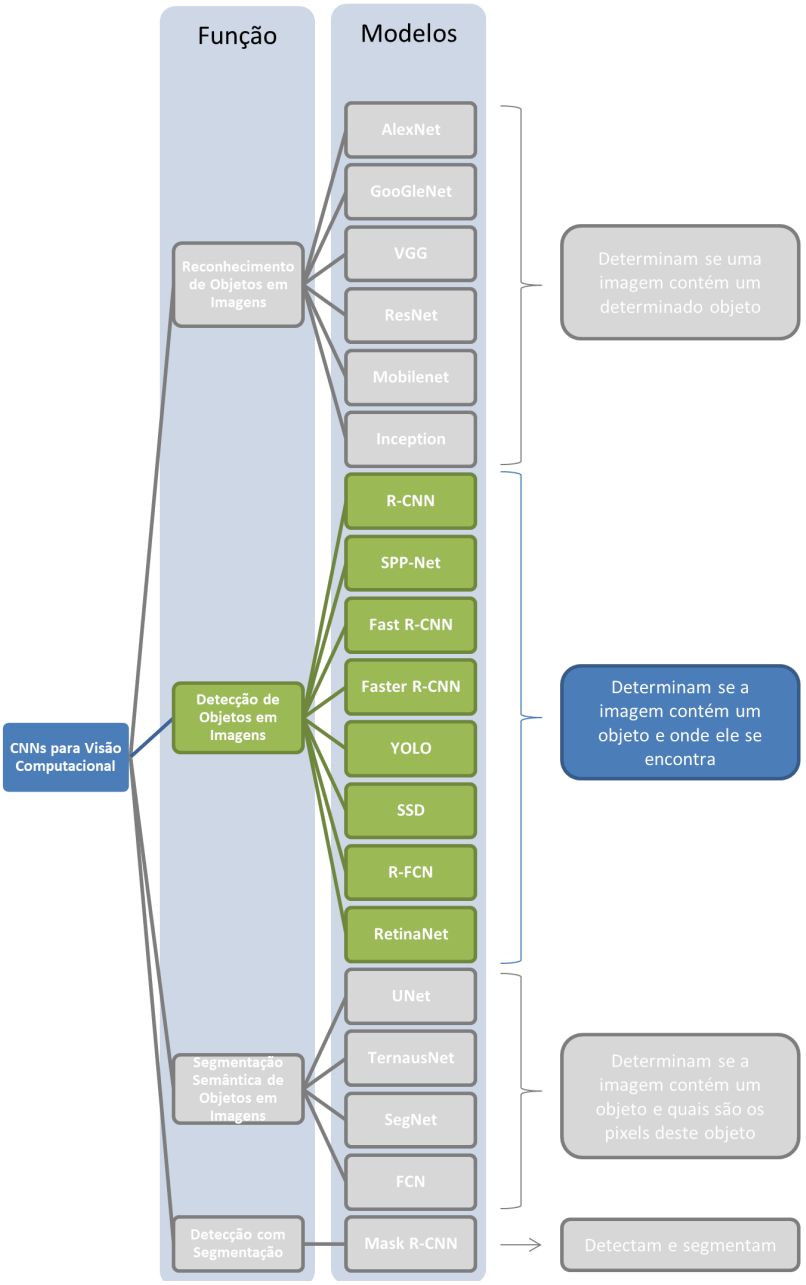
Dada a natureza desses requisitos que o modelo deve atender, existe uma série de abordagens possíveis documentadas na literatura para tarefa de detecção de elementos em IUs, que vão desde o campo da visão computacional clássica até o campo de *machine learning*, quais sejam: 1. Detecção de Borda, 2. Correspondência de Template, 3. CNNs de Dois Passos, 4. CNNs de Um Passo e 5. Abordagem Híbrida com Métodos Clássicos e CNNs (Degaki et al., 2022), sendo 1. e 2. abordagens clássicas e 3., 4. e 5. abordagens com *machine learning*.

A Detecção de Borda é um método clássico que utiliza a identificação de bordas e regiões primitivas para formar janelas ou objetos maiores, com a classificação feita heurísticamente. Já a Correspondência de Template depende de atributos manuais e é eficaz apenas para IUs simples. Em CNNs de Dois Passos, duas redes são treinadas, uma para detectar regiões de interesse (passo 1) e outra para classificar os objetos (passo 2), como no método FasterRCNN. As CNNs de um Passo são mais rápidas e inferem localização e classe dos objetos simultaneamente, sendo ideais para aplicações em tempo real. Finalmente, métodos híbridos combinam técnicas clássicas com CNNs para corrigir a precisão da localização dos objetos.

Considerando-se os requisitos do modelo deste trabalho, e o melhor desempenho das abordagens de *machine learning* em comparação com as clássicas de visão computacional (Krizhevsky et al., 2012), optou-se neste trabalho pelo uso de CNNs, que são ideias para detecção de objetos, pois são altamente eficientes em aprender e identificar padrões espaciais em imagens. Isso porque, utilizando camadas convolucionais, as CNNs extraem características de diferentes

escalas e localizações da imagem, como bordas, texturas e formas, sem precisar de intervenção manual. Isso permite que as CNNs reconheçam objetos com alta precisão, mesmo em condições variadas, como diferentes ângulos, tamanhos ou iluminações. Além disso, a capacidade de generalização das CNNs as torna adequadas para tarefas complexas de detecção de objetos em imagens reais (Girshick et al., 2014).

Figura 10 – CNNs para visão computacional



Fonte: (Wangenheim et al., 2018)

No entanto, como se pode observar na Figura 10, existem diversos tipos de CNNs para detecção de objetos, sendo elas divididas, como mencionado anteriormente, entre CNNs de Dois Passos e CNNs de Um passo. Nesse cenário, foi selecionado para a construção do modelo extração semântica o modelo YOLO (*You Only Look Once*), que como o próprio nome sugere é um modelo de um passo.

A escolha do modelo YOLO para a detecção de objetos se justifica principalmente pela sua combinação eficiente de alta precisão e velocidade de inferência. Ao contrário dos modelos de dois estágios, como o Faster R-CNN, que primeiro geram propostas de regiões e depois realizam a classificação dessas regiões, o YOLO realiza a detecção de objetos em uma única passagem pela rede, o que resulta em um desempenho significativamente mais rápido. Isso é especialmente importante em aplicações em tempo real, como é esperado no caso da ferramenta CodeMaster.

Estudos como os de Redmon e Farhadi (2018) e Wangenheim et al. (2018), mostram que o modelo não só oferece inferências rápidas, mas também alcança resultados comparáveis ou até superiores a outros modelos em termos de precisão. Portanto, a escolha do YOLO para tarefas de detecção de objetos, em comparação com modelos de dois estágios, é justificada pela sua eficácia em alcançar um bom equilíbrio entre precisão e velocidade, tornando-o ideal no presente contexto.

4.3 CONJUNTO DE DADOS

Um conjunto de dados de treino para uma CNN YOLO tem por dados imagens e um arquivo de texto associado a cada uma das imagens contendo a posição e classificação dos objetos na imagem. Assim, para a criação desse conjunto, decidiu-se utilizar o RICO (Deka et al., 2017), um dos maiores conjuntos de dados e metadados de aplicativos Android, contendo 66261 IUs distintas de 9772 aplicativos. Além de conter os *screenshots* (em PNG) e a hierarquia de elementos (em JSON) de cada IU com anotações semânticas dos elementos. As anotações semânticas do RICO fornecem informações sobre os diferentes elementos das interfaces, como botões, campos de texto, menus e imagens, identificando tanto a classe do objeto quanto suas coordenadas em relação à tela. Nesse conjunto de dados também são disponibilizados um conjunto de interações do usuário com a IU (JSON), um conjunto de animações da captura de efeitos de transição em resposta às interações

do usuário (GIF) e uma representação vetorial aprendida de seu layout (JSON e NPY).

Como os arquivos de metadados JSON do RICO não seguem o formato do arquivo de texto esperado por um modelo YOLO (Figura 11), foi desenvolvida, no contexto deste trabalho, uma ferramenta *ad hoc* (escrita em linguagem de programação Java) capaz de inspecionar os arquivos de hierarquia de elementos com anotações semânticas do RICO e gerar a partir deles arquivos que seguem o padrão de entrada de dado dessa CNN (Porto, 2024).

Figura 11 – Formato do arquivo de texto para um modelo YOLO

```
[category number] [object center in X] [object center in Y] [object
width in X] [object width in Y]
```

Fonte: (Enrique, 2018).

A partir desse formato, é gerado um arquivo de texto para cada imagem de IU, onde cada linha descreve um objeto cujas propriedades podem ser definidas como:

- *Category number*: um número inteiro que representa a categoria do elemento de IU, conforme o arquivo de anotações semânticas do RICO (Quadro 13);
- Coordenadas normalizadas do objeto em relação ao tamanho da imagem (dadas em valores relativos e não absolutos) por meio das propriedades:
    - *object center in X:* posição relativa do centro do objeto em relação à largura da imagem;
    - *object center in Y:* posição relativa do centro do objeto em relação à altura da imagem;
    - *object width in X:* tamanho relativo da largura do objeto em relação à largura da imagem;
    - *object height in Y:* tamanho relativo da altura do objeto em relação à altura da imagem.

Quadro 13 – Categorias usadas para detecção de elementos em IUs

Categoria	ID
Advertisement	0

Background Image	1
Bottom Navigation	2
Button Bar	3
Card	4
Checkbox	5
Drawer	6
Date Picker	7
Image	8
Image Button	9
Input	10
List Item	11
Map View	12
Multi-Tab	13
Number Stepper	14
On/Off Switch	15
Pager Indicator	16
Radio Button	17
Slider	18
Text Button	19
Toolbar	20
Video	21
Web View	22
Text	23
Icon	24
Modal	25

Fonte: elaborado pelo autor

Além disso, para garantir a qualidade e a generalização do modelo, o conjunto de dados foi dividido (Hugging Face, 2020) em três partes principais:

1. Treinamento (80%): a maior parte do conjunto de dados foi alocada para treinamento, permitindo que o modelo aprenda a identificar e classificar diferentes elementos das interfaces. O treinamento é onde o modelo ajusta seus pesos e aprende a partir dos exemplos fornecidos.
2. Validação (10%): o conjunto de validação foi usado durante o treinamento para monitorar o desempenho do modelo em dados que ele não viu antes. Isso ajuda a evitar o *overfitting* e permite ajustar os hiperparâmetros do modelo, como a taxa de aprendizado e o número de épocas. O desempenho no conjunto de validação é uma métrica crucial para determinar se o modelo está generalizando bem.
3. Teste (10%): após o treinamento, o conjunto de teste foi utilizado para avaliar o modelo final. Este conjunto contém dados que o modelo nunca viu e serve para fornecer uma estimativa realista da performance do modelo em dados não vistos. É aqui que as métricas como precisão, *recall* e mAP são calculadas para verificar a qualidade final da detecção.

Essa divisão dos dados em treinamento, validação e teste é fundamental para garantir a robustez e a generalização do modelo.

4.4 TREINAMENTO

Objetivando extrair deste modelo de CNN o máximo de recursos e desempenho, utilizou-se a implementação de YOLO da Ultralytics em sua versão 11 (Ultralytics, 2024), sendo essa a mais nova disponível no momento de execução deste trabalho.

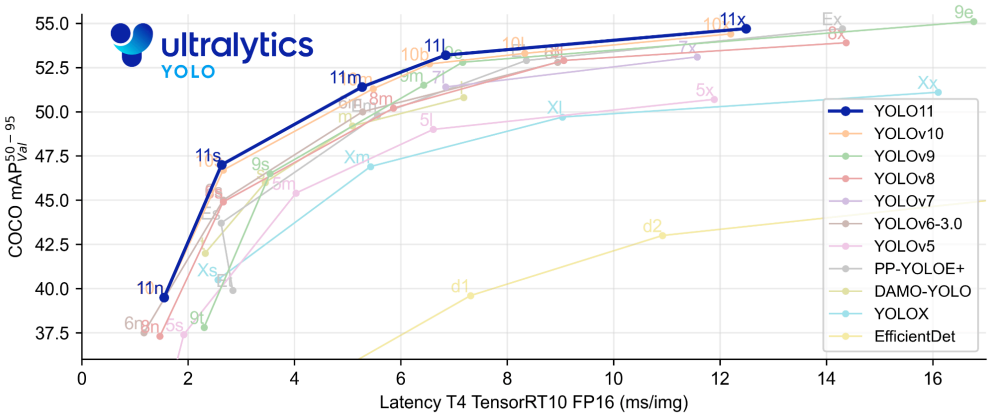
Além disso, para otimizar o treinamento, foi utilizada uma técnica de treinamento de RNAs chamada de *transfer learning*, que consiste em utilizar o conhecimento previamente adquirido por um modelo em um conjunto de dados para acelerar o treinamento e melhorar o desempenho com um novo conjunto de dados (Pan e Yang, 2010). O cerne dessa técnica é reaproveitar os parâmetros aprendidos por um modelo treinado em um grande conjunto de dados para adaptar-se a um problema mais específico com menos dados ou recursos computacionais (Goodfellow et al., 2016).

A aplicação de *transfer learning* no contexto deste trabalho idealmente se daria pelo uso de um modelo pré-treinado na tarefa de detecção de elementos em UIs, no entanto, dada a escassez de tais modelos (Degaki et al., 2022), foi utilizado um modelo pré-treinado mais genérico fornecido pela implementação de YOLO da Ultralytics.

Esse modelo foi treinado com base no conjunto de dados COCO (Common Objects in Context), um dos mais utilizados para tarefas de detecção de objetos, contendo uma grande variedade de imagens de diferentes categorias de objetos e, embora não seja especificamente voltado para IUs, oferece uma base para *transfer learning* sobre características visuais básicas, como bordas, texturas e formas. Assim, essas características podem ser reaproveitadas para detecção de elementos em IUs, adaptando as camadas finais do modelo para identificar elementos de IUs mais específicos.

Por fim, a implementação Ultralytics disponibiliza diferentes versões de modelos pré-treinados em COCO, sendo elas: *nano* (n), *small* (s), *medium* (m), *large* (l) e *extreme large* (x). Cada uma possui diferentes tempos de resposta e eficiência (Figura 12).

Figura 12 – Desempenho de modelos pré-treinados em COCO da Ultralytics



Fonte: (Ultralytics, 2024)

A versão mais básica, *nano*, apresenta o melhor tempo de resposta, mas a menor eficiência. Por outro lado, a versão mais complexa, *extreme large*, apresenta a maior eficiência, mas também o pior tempo de resposta. Assim, optou-se por

utilizar para *transfer learning* a versão *small*, objetivando um equilíbrio adequado entre tempo de resposta e eficiência do modelo pré-treinado.

4.4.1 Hardware

O treinamento de RNAs profundas consiste em um tipo de processamento computacional que consome muitos recursos de *hardware*, particularmente de GPUs (*Graphics Processing Units*).

As GPUs são importantes para os treinamentos de RNAs porque processam uma grande quantidade de cálculos de forma paralelizada, como multiplicações de matrizes, sendo essas multiplicações fundamentais para esse tipo de processamento (Goodfellow et al., 2016). Assim, a escolha de uma boa GPU é essencial para um treinamento mais eficiente.

Dado que o treinamento foi realizado no ambiente Google Colab (Google, 2024), que oferece 3 modelos de GPUs distintas (NVIDIAs L4, T4 e A100), optou-se pelo uso da GPU NVIDIA A100, uma das GPUs mais potentes disponíveis para treinamento de redes neurais, oferecendo aceleração de alto desempenho para modelos de *deep learning* (Nvidia, 2024).

Assim, a escolha da A100 foi importante para acelerar o processo de treinamento, tendo em vista o tamanho expressivo do conjunto de dados, o objetivo de criação de um modelo performático e a necessidade de tempos de treinamento razoáveis ao contexto deste trabalho.

4.4.2 Hiperparâmetros

Dada a importância dos hiperparâmetros para o desempenho do modelo, eles foram cuidadosamente escolhidos com base em boas práticas e na literatura recente. A seguir, estão os principais hiperparâmetros utilizados no treinamento.

- ***Epochs* (épocas):** é o número de vezes com que a RNA faz uma varredura completa no conjunto de dados para consolidar o aprendizado. A escolha desse valor necessita de um bom equilíbrio, pois um número baixo de épocas pode ser insuficiente para que o modelo apresente bom aprendizado, mas um número elevado também pode ocasionar *overfitting.* Um limite de 50 épocas foi inicialmente planejado. Durante o treinamento, a métrica de mAP@0.5 foi monitorada, e o treinamento foi interrompido antecipadamente caso a

validação não apresentasse melhorias nas últimas 10 épocas, prática comum para evitar o *overfitting*. Esse número de épocas foi escolhido com base na literatura sobre YOLO, onde geralmente, para grandes conjuntos de dados, são necessárias entre 50 e 100 épocas para alcançar boa convergência (Redmon et al., 2016).

- ***Learning Rate* (taxa de aprendizado)**: é a taxa com que os pesos da RNA são atualizados em cada iteração do treinamento. Ou seja, o tamanho do passo que RNA dá para aprender. Um valor alto possibilita um aprendizado mais rápido, mas que pode não criar um modelo preciso e um valor baixo tende a criar um modelo mais preciso, mas que precisa de mais tempo para ser treinado. Dessa forma, foi utilizado o valor de 0.01 com uma estratégia de decaimento exponencial, começando então em um valor moderado, evitando grandes oscilações nos pesos e garantindo uma convergência estável. Com o uso do decaimento garante-se a redução da taxa de aprendizado ao longo das épocas, ajudando a refinar a aprendizagem nas últimas fases.
- ***Batch Size* (tamanho de lote):** é o número de dados do conjuntos de dados que é processado de uma única vez no treinamento da RNA para ajustar seus pesos, utilizando-se da média do erro desse subconjunto para o ajuste dos pesos. É um parâmetro crítico, já que valores maiores podem acelerar o treinamento, mas exigem maior capacidade de memória. 64 foi escolhido como valor, levando em consideração a memória da GPU NVIDIA A100, sendo um bom equilíbrio entre eficiência e uso da memória.
- ***Momentum*:** é usado para que o ajuste dos pesos da RNA não se dê somente com base no gradiente da época corrente, mas também com base nos ajustes das épocas anteriores, o que na prática acelera o treinamento e ajuda a rede a escapar de mínimos locais. 0.937 foi utilizado, um valor comumente adotado em treinamentos de modelos YOLO.
- ***Input Size* (tamanho de entrada)**: refere-se à dimensão das imagens que a rede é treinada para processar. 448 x 448 pixels foi a resolução escolhida para as imagens de entrada. Embora uma resolução maior possa aumentar a precisão, ela também demanda mais recursos computacionais. O valor de 448 x 448 foi escolhido por ser um bom equilíbrio entre desempenho e recursos, conforme recomendado por Redmon et al. (2016).

4.4.3 Estratégias de Prevenção de *Overfitting*

Overfitting é um problema comum em *machine learning*, que consiste em um modelo treinado de forma muito acoplada ao conjunto de dados de treinamento, fazendo com que ele memorize os padrões específicos desses dados, incluindo ruídos ou características irrelevantes. Desse modo, o modelo perde a capacidade de generalização quando é apresentado a novos dados, resultando em um desempenho ruim nos conjuntos de validação e teste, o que, em um última instância, se traduz em um modelo que não apresentará bons resultados ao usuário final.

Para evitar *overfitting*, foram adotadas as seguintes estratégias durante o treinamento:

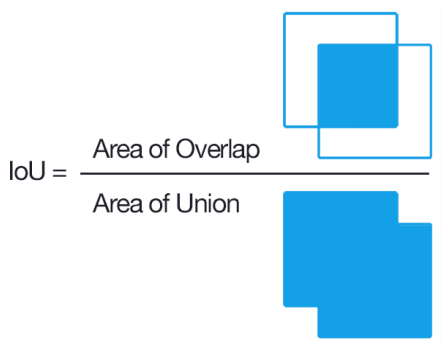
- **Regularização**: foram utilizadas técnicas de *dropout* e *data augmentation*.
    - ***Dropout*** funciona desativando aleatoriamente uma porcentagem de neurônios da RNA em cada iteração do treinamento, forçando o modelo a aprender representações mais robustas e a não depender excessivamente de neurônios específicos.
    - Já ***data augmentation*** consiste no aumento artificial (por meio de transformações como rotação, zoom e *flip* horizontal) para aumentar a variabilidade dos dados de entrada e melhorar a capacidade de generalização do modelo.
- **Early Stopping**: o treinamento foi interrompido antecipadamente quando o desempenho na validação não evoluiu após 10 épocas consecutivas.

4.5 INFERÊNCIA

Foram utilizadas diferentes métricas objetivando acompanhar o treinamento e avaliar o desempenho do modelo treinado, sendo as principais a *Intersection over Union* (IoU) e o *F1 score*.

A IoU define o quanto o objeto detectado se aproxima do objeto real (Zou et al., 2019), medindo a sobreposição entre a área prevista pelo modelo (*bounding box* predito) e a área real anotada (*bounding box* verdadeiro), conforme indica a Figura 13.

Figura 13 – Cálculo da métrica *Intersection over Union*



Fonte: (Rosebrock, 2016)

É por meio dessa métrica que se deriva outras métricas que são utilizadas para avaliar o desempenho do modelo como as métricas de *precision*, *recall*, *mean average* e *mean average precision*, que são definidas da seguinte forma:

1. *Precision*: mede a proporção de detecções corretas (verdadeiros positivos) em relação ao total de detecções realizadas pelo modelo (total positivos), indicando a exatidão do modelo ao evitar falsas detecções. Quanto maior a *precision*, melhor.
2. *Recall*: mede a proporção de detecções corretas (verdadeiros positivos) em relação ao total de elementos existentes (relevantes). Essa métrica indica a capacidade do modelo de encontrar todos os elementos relevantes. Quanto maior o *recall*, melhor.
3. *Average precision*: combina *precision* e *recall* para avaliar a qualidade do modelo em identificar e localizar corretamente cada categoria de elemento dentro das imagens. Quanto maior a *average precision*, melhor.
4. *Mean Average Precision* (mAP): mede a precisão geral do modelo em identificar e localizar corretamente objetos de diferentes classes, com base na *average precision* de cada uma das classes. Quanto maior a mAP, melhor.
    a. mAP@0.5: neste cálculo de mAP a predição é considerada correta se a área de sobreposição entre a *bounding box* predita e a real for pelo menos 50%, ou seja, se IoU >= 0.50. É uma métrica mais permissiva, focada em capturar a capacidade geral do modelo de localizar e classificar objetos corretamente.
    b. mAP@0.95: neste cálculo de mAP a predição é considerada correta se a área de sobreposição entre a *bounding box* predita e a real for pelo

menos 95%, ou seja, se IoU >= 0.95. Essa métrica é mais rigorosa, pois avalia o modelo em diferentes critérios de precisão espacial. Assim, ela reflete melhor a capacidade do modelo de localizar objetos com alta precisão.

Por sua vez, o *F1 score* é a média harmônica da *precision* e *recall* (Robinson, 2019), transmitindo informações sobre o equilíbrio entre essas métricas, com um alto *F1 score* indicando que o sistema executa uma detecção que é exata e completa (Robinson, 2019).

A partir do acompanhamento dessas métricas, foi identificado que o modelo alcançou seu melhor desempenho após **16000 *batches***, ponto em que as métricas de *precision* e *recall* se mostraram equilibradas. O modelo, nessa fase, apresentou uma **mAP@0.5 de 81.56%, mAP@0.95 de 67.41%** e um ***F1 score* de 76.17%**, que são valores que representam um modelo eficiente na detecção e com um ponto de equilíbrio entre *precision* e *recall*, não indicando *overfitting*. Além disso, esses valores indicam também uma alta precisão na detecção dos elementos de IU, mantendo uma taxa satisfatória de detecção correta entre todos os objetos relevantes presentes.

4.6 EXPORTAÇÃO

Finalmente, o modelo produzido neste trabalho foi exportado (Porto, 2024) para os formatos ONNX (Open Neural Network Exchange) (Onnx, 2024) e TensorRT (Nvidia, 2024).

A exportação para esses formatos amplia a aplicabilidade e eficiência do modelo, facilitando seu uso em dispositivos variados e aplicações que demandam alto desempenho, tornando assim viável a sua integração em diferentes ambientes e ferramentas, incluindo o CodeMaster.

5. AVALIAÇÃO DO MODELO “TifSX”

A avaliação do modelo foi conduzida com o propósito de analisar os seguintes aspectos de qualidade:

OA1. Correspondência entre o resultado da detecção e os elementos das IUs do App Inventor;
OA2. Tempo de resposta do modelo para detectar elementos nas IUs do App Inventor.

Para avaliar o modelo com base nos objetivos de avaliação descritos, seguiu-se um método de avaliação que considera as melhores práticas de análise de modelos de detecção de objetos usando redes neurais convolucionais (CNNs) e as métricas padrão em visão computacional (Zhao et al, 2018). Esse método é dividido em duas partes: a avaliação da precisão de correspondência (OA1) e a medição do tempo de resposta (OA2).

5.1 AVALIAÇÃO DA CORRESPONDÊNCIA ENTRE O RESULTADO DA DETECÇÃO E OS ELEMENTOS

Para avaliar a performance do modelo de detecção de elementos de IUs em condições realistas e adequadas à ferramenta CodeMaster, foi preparado um conjunto de dados exclusivo para teste, composto por 100 IUs de aplicativos desenvolvidos com o App Inventor.

A coleta das imagens de IUs do App Inventor se deu pelo uso de um *script* desenvolvido para a ferramenta AutoIt (Autoit, 2024), que permitiu uma coleta semi-automatizada de *screenshots* de aplicativos da Galeria do App Inventor (App MIT, 2019).

Uma vez coletadas as 100 imagens de IUs, todos os elementos presentes nas IUs foram rotulados de forma manual por meio da ferramenta YoloMark (Alexey, 2024), conforme os mesmos elementos de IU que foram utilizados no conjunto de dados de treinamento.

Esse conjunto de dados de teste é completamente novo, ou seja, não foi utilizado durante a fase de treinamento do modelo. Esse isolamento do conjunto de teste é uma prática recomendada em aprendizado de máquina e visão computacional, pois evita o fenômeno de overfitting e garante que o modelo seja validado em dados inéditos, representando melhor sua capacidade de generalização

e robustez ao detectar objetos em novos contextos de interface (Goodfellow et al., 2016).

Com isso, para garantir que o modelo detecta e classifica corretamente os elementos da IU de aplicativos do App Inventor presentes nesse conjunto de dados, foram utilizadas, mais uma vez, as métricas *Intersection over Union* (com as métricas derivadas de mAP) e *F1 score*, de modo que os resultados obtidos para o modelo são apresentados na Tabela 3.

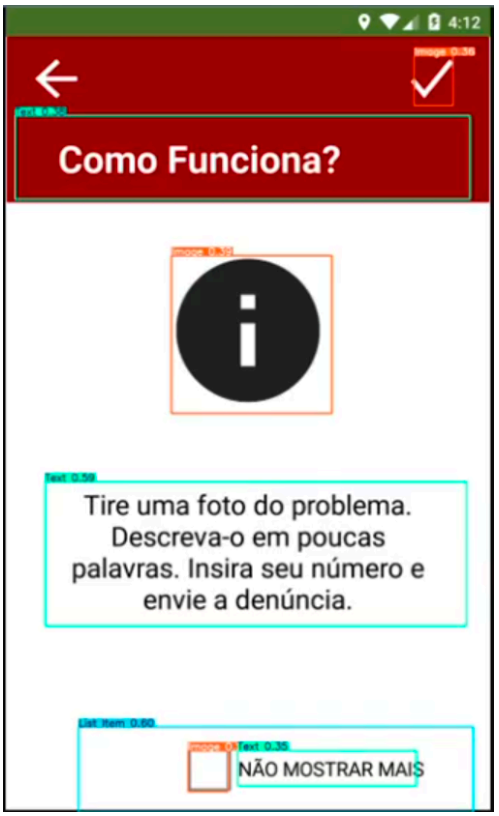
Tabela 3 – Resultados obtidos na avaliação do modelo “TifSX”

Métrica	Resultado
mAP@0.5	0.7240
mAP@.95	0.5384
F1 score	0.7116

Fonte: elaborado pelo autor

Os resultados obtidos para o TifSX indicam um desempenho razoável, mas com áreas que podem ser aprimoradas. O mAP@0.5 de 72.40% é relativamente bom, indicando que 72,4% das detecções feitas pelo modelo são corretas quando se considera IoU >= 50%, o que é um valor aceitável em modelos de detecção de objetos (Redmon et al., 2016). No entanto, o mAP@.95 de 53.84% sugere que o modelo tem dificuldade em atender aos critérios mais rigorosos de localização e classificação (Lin et al., 2014). Por fim, o *F1 score* de 71.16% está alinhado aos valores de mAP, mostrando que há um equilíbrio razoável entre *precision* (evitar falsos positivos) e *recall* (evitar falsos negativos). No entanto, apesar de valores acima de 70% na métrica de *F1 score* serem considerados bons (Robinson, 2019), há espaço significativo para melhorias no ajuste fino do modelo.

Figura 14 – Exemplo da detecção de elementos de uma IU usando TifSX



Fonte: elaborado pelo autor

Em termos de detecção (Figura 14), esses resultados são compatíveis com o que seria esperado de um modelo YOLO (Redmon et al., 2016) com um bom equilíbrio entre velocidade e precisão, mas com áreas claras a melhorar, especialmente na capacidade de detectar e localizar corretamente os objetos.

5.2 AVALIAÇÃO DO TEMPO DE RESPOSTA DA DETECÇÃO DO MODELO

O tempo de resposta é essencial para avaliar o desempenho do modelo em tempo real, especialmente para cenários que exigem respostas rápidas do modelo, como é o caso de avaliações feitas pela ferramenta CodeMaster. Para essa avaliação, considera-se o tempo médio de inferência por imagem, utilizando o conjunto de dados de 100 IUs do App Inventor.

Constatou-se com as inferências a partir desse conjunto de dados um tempo médio de 63 ms para predição de cada imagem, o que é um tempo razoável no contexto da avaliação automatizada feita pela ferramenta CodeMaster, não

impactando o tempo de resposta, e portanto a usabilidade, dessa ferramenta a partir da integração com o TifSX.

5.3 DISCUSSÃO

O modelo TifSX desenvolvido demonstrou por meio de suas métricas um desempenho razoável, mas com possibilidades para melhorias, que incluem:

- **Melhoria das métricas de mAP e *F1 score***: as métricas obtidas, apesar de boas, podem ser otimizadas. Isso pode se dar por diferentes ações, como a exploração de novos valores de hiperparâmetros (como *learning rate*, *batch size*, *input size* etc), o uso de modelos pré-treinados mais robustos para aplicação de *transfer learning* ou a expansão do conjunto de dados.
- **Melhoria na quantidade de semântica extraída:** as 26 categorias de elementos de IU já provêm uma quantidade de semântica razoável, mas podem ser expandidas pela exploração de mais dados das anotações semânticas do conjunto RICO. Isso exige, no entanto, *hardware* com maior poder computacional para treinamento.

Apesar dessas melhorias, o TifSX superou as métricas de precisão de modelos focados na detecção de elementos em IUs previamente descritos na literatura (Degaki et al., 2022), refletindo a capacidade do modelo em identificar corretamente elementos das IUs sem incorrer em um número elevado de falsos positivos. Esse desempenho implica na consolidação do uso de modelos baseados em YOLO como uma abordagem eficiente para detecção de elementos em IUs.

A partir disso, deriva-se a viabilidade do uso do modelo desenvolvido para extração da semântica de *design* de elementos de IU de aplicativos do App Inventor, principal objetivo deste trabalho, por meio da integração do modelo à ferramenta CodeMaster. Além disso, como quando comparado a métodos clássicos da visão computacional, o modelo oferece uma solução moderna e aplicável em *hardware* limitado, dado seu tempo de resposta baixo, ele mostra-se especialmente vantajoso para a integração em ambientes com restrições computacionais, como tendem a ser os ambientes de Educação Básica.

7. CONCLUSÃO

A partir dos resultados obtidos com o modelo TifSX, desenvolvido para detecção de objetos em interfaces de usuário de aplicativos Android criados no App Inventor, pode-se concluir que o modelo apresenta desempenho satisfatório, com métricas de avaliação adequadas aos padrões esperados em arquiteturas de detecção do tipo YOLO. As métricas de desempenho, como mAP baseada em *Intersection over Union*, indicam que o modelo é capaz de identificar e classificar componentes de IUs com uma precisão adequada, atendendo aos requisitos básicos de um sistema de análise de *design* de interface e superando o desempenho de outros modelos anteriormente propostos na literatura para tarefas similares, demonstrando sua eficácia e relevância no domínio específico de detecção de elementos em interfaces gráficas.

O modelo mostrou-se, assim, pronto para ser integrado à ferramenta CodeMaster, ampliando as capacidades desta na avaliação de usabilidade e *design* em IUs criadas por estudantes de computação. Essa integração potencializará o processo de ensino-aprendizagem, permitindo que alunos possam receber *feedback* automatizado sobre o *design* de seus aplicativos, incentivando a melhoria contínua. Assim, espera-se que a combinação do modelo TifSX com o CodeMaster traga avanços significativos para a área de avaliação de usabilidade e *design* de interfaces no contexto educacional, contribuindo para o desenvolvimento de competências essenciais em *design* e programação na Educação Básica.

No entanto, apesar dos avanços alcançados, o trabalho apresenta algumas limitações. A integração direta do modelo com a ferramenta CodeMaster, prevista inicialmente, não foi concluída no escopo deste projeto. Essa etapa foi substituída pela disponibilização do modelo em diferentes formatos, como ONNX e TensorRT, permitindo que desenvolvedores interessados façam a integração futuramente. Além disso, os valores de métricas obtidos apresentam margens para melhorias. Assim, trabalhos futuros incluem 1) a efetiva integração do modelo com o CodeMaster, criando uma *pipeline* automática que permita a análise em tempo real de IUs criadas por estudantes e 2) a realização de *fine-tuning* do modelo explorando diferentes valores de hiperparâmetros e um conjunto de dados mais amplo e diversificado.

REFERÊNCIAS

Alexey. “AlexeyAB/Yolo_mark”. GitHub. Disponível em: <https://github.com/AlexeyAB/Yolo_mark>.

Alves, N. D. C., Wangenheim, C. G. V., Hauck, J. C. R., “Approaches to Assess Computational Thinking Competences Based on Code Analysis in K-12 Education: A Systematic Mapping Study”. Informatics in Education, in press, 2019.

AutoIt. “AutoIt Downloads”. AutoIt. Disponível em: <https://www.autoitscript.com/site/autoit/downloads/>. Acesso em: 5 dez. 2024.

CSTA, “Standards - CSTA K-12 Computer Science Standards” (2017). Disponível em: <https://www.csteachers.org/page/standards>. Acesso em: 12 abr. 2019.

Data Science Academy, “Uma Breve História das Redes Neurais Artificiais” (2019). Disponível em: <http://deeplearningbook.com.br/uma-breve-historia-das-redes-neurais-artificiais/>. Acesso em: 12 abr. 2019.

Degaki R. H., Colonna J. G.; Lopez, Y. et al. “Real Time Detection of Mobile Graphical User Interface Elements Using Convolutional Neural Networks”. 2022.

Deniz Eseryel, Dirk Ifenthaler and Xun Ge (2013). Validation study of a method for assessing complex ill-structured problem solving by using causal representations. Educational Technology Research. vol. 61. p. 443–463.

Deka, B., Huang, Z.; Franzen, C. et al. “Rico: A Mobile App Dataset for Building Data-Driven Design Applications”. Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, 2017.

Enrique A., “Object detection with YOLO: implementations and how to use them” (2018). Disponível em: <https://medium.com/@enriqueav/object-detection-with-yolo-implementations-and-how-to-use-them-5da928356035>. Acesso em: 12 abr. 2019.

Ferreira, M. N. F., Wangenheim, C. G. V., Missfeldt Filho, R., Pinheiro, F. D. C, Hauck, J. C. R., “Learning user interface design and development of mobile applications in middle school”. ACM Interactions, in press, 2019.

Girshick, R.; Donahue, J.; Darrell, T. et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. 2014 IEEE Conference on Computer Vision and Pattern Recognition, p. 580–587, 2014. Disponível em: <https://dl.acm.org/citation.cfm?id=2679851>.

Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. MIT Press.

Google. “colab.google”. colab.google. Disponível em: <https://colab.google/>. Acesso em: 5 dez. 2024.

Google Design, “Material Design” (2019). Disponível em: <https://material.io/design/>. Acesso em: 12 abr. 2019.

Grover, S. e Pea, R, “Computational Thinking in K–12: A Review of the State of the Field” (2013), Educational Researcher, 42(1), 38–43. <doi:10.3102/0013189x12463051>.

Kartinah Zen, D.N.F Awang Iskandar and Ongkir Linang (2011). Using Latent Semantic Analysis for automated grading programming assignments. Proc. of the Int. Conference on Semantic Technology and Information Retrieval, Kuala Lumpur, Malaysia.

Kepios Pte. Ltd., “Digital 2019: Brazil – DataReportal – Global Digital Insights (2019). Disponível em: <https://datareportal.com/reports/digital-2019-brazil>. Acesso em: 12 abr. 2019.

Kierski, M., “Machine Learning development process - you’ve got it wrong” (2017). Disponível em: <https://medium.com/sigmoidal/machine-learning-development-process-youve-got-it-wrong-396270e653f4>. Acesso em: 12 abr. 2019.

Kirsti Ala-Mutka and Hannu-Matti Järvinen. (2004). Assessment process for programming assignments. Proc. of IEEE Int. Conference on Advanced Learning Technologies, Joensuu, Finland.

Krizhevsky, A., Sutskever, I., Hinton, G. E., “ImageNet classification with deep convolutional neural networks”. Communications Of The Acm, [S.L.], v. 60, n. 6, p. 84-90, 24 maio 2017. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/3065386.

Lecun, Y., Bottou, L., Bengio, Y., et al. “Gradient-based learning applied to document recognition”. Proceedings of the IEEE, v. 86, n. 11, p. 2278–2324, 1998.

Liu, T. F. et al, “Learning Design Semantics for Mobile Apps”. The 31st Annual Acm Symposium On User Interface Software And Technology - Uist '18, [s.l.], p.569-579, 2018. ACM Press. <http://dx.doi.org/10.1145/3242587.3242650>.

MIT, “MIT App Inventor News & Events | Explore MIT App Inventor” (2019). Disponível em: <http://appinventor.mit.edu/>. Acesso em: 12 abr. 2019.

MORE: Mecanismo online para referências, versão 2.0. Florianópolis: UFSC Rexlab, 2013. Disponível em: http://www.more.ufsc.br/. Acesso em: 18 nov. 2024.

B. Myers, “Challenges of hci design and implementation,” Interactions, vol. 1, no. 1, pp. 73–83, Jan. 1994. [Online].

Nielsen, J., “Usability Engineering. Morgan Kaufmann” (1993), 362 p.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., “Systematic Mapping Studies in Software Engineering”. Proc. of the 12th international conference on Evaluation and Assessment in Software Engineering, Bari, Italy, 2008, p. 68-77.

Polyzotis, N., Roy, S., Whang, S. E., Zienkevich, M., “Data Management in Production Machine Learning”. Proc. of the ACM International Conference on Management of Data, Chicago, IL, USA, 2017.

Porto, J. V. A., Barbosa, H., Wangenheim, C. G. V., “Proposta de um Checklist de Avaliação de Usabilidade de Aplicativos Android no Contexto Educacional”. In: Computer on the Beach, 2018, Florianópolis.

Redmon, J., Farhadi, A., “YOLOv3: An Incremental Improvement”. arXiv.org. Disponível em: <https://arxiv.org/abs/1804.02767>.

Redmon, J.; Divvala, S.; Girshick, R. et al. “You Only Look Once: Unified, Real-Time Object Detection”. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), p. 779–788, 2016. Disponível em: <https://arxiv.org/pdf/1506.02640.pdf>.

Robinson, A. “Sketch2code: Generating a website from a paper mockup”, 2019. Disponível em: <https://arxiv.org/abs/1905.13750>. Acesso em: novembro 2019.

Rosebrock A.. “Intersection over Union (IoU) for object detection”. PyImageSearch. Disponível em: <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>.

Shuai, “Steps to Follow in Deep Learning Projects” (2017). Disponível em: <https://shuaiw.github.io/2017/09/27/steps-to-follow-in-deep-learning-projects.html>. Acesso em: 12 abr. 2019.

C. Szegedy, A. Toshev, and D. Erhan, “Deep neural networks for object detection,” in Proc. Adv. Neural Inf. Process. Syst. Conf., 2013, pp. 2553–2561.

Balaban, Stephen. “Deep Learning and Face Recognition: The State of the Art.” Ed. Ioannis A. Kakadiaris, Ajay Kumar, and Walter J. Scheirer. Biometric and Surveillance Technology for Human and Activity Identification XII (2015): n. pag. Crossref. Web.

Pan, S. J., Yang, Q.. “A Survey on Transfer Learning”. IEEE Transactions on Knowledge and Data Engineering, v. 22, n. 10, p. 1345–1359, 2010.

Petersen, K. et al. (2008) Systematic mapping studies in software engineering. In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, 68-77.

Porto, J. V. A., “TifSX”. GitHub. Disponível em: <https://github.com/joaovaporto/TifSX>. Acesso em: 23 dez. 2024.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. IEEE Computer Vision and Pattern Recognition (CVPR), 2009.

Haddaway NR, Collins AM, Coughlin D, Kirk S. The role of Google Scholar in evidence reviews and its applicability to Grey literature searching. PLoS One 2015;10:e0138237.

K. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk. “Machine learning-based prototyping of graphical user interfaces for mobile apps”. IEEE Transactions on Software Engineering, 2018.

Nvidia. “NVIDIA A100 GPUs Power the Modern Data Center”. NVIDIA. Disponível em: <https://www.nvidia.com/en-us/data-center/a100/>. Acesso em: 5 dez. 2024.

ONNX. “Introduction to ONNX - ONNX 1.17.0 documentation”. onnx.ai. Disponível em: <https://onnx.ai/onnx/intro/>. Acesso em: 5 dez. 2024.

S. Chen, L. Fan, T. Su, L. Ma, Y. Liu, and L. Xu. “Automated cross-platform GUI code generation for mobile apps”. In Proc. of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER. IEEE, 2019.

Shiri F. M., Perumal T., Mustapha N. et al. “A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU”. arXiv.org. Disponível em: <https://arxiv.org/abs/2305.17473>.

Shuchi Grover, Stephen Cooper and Roy Pea (2014). Assessing Computational Learning in K-12. Proc. of the Conference on Innovation & Technology in Computer Science Education, Uppsala, Sweden.

Solecki, I.; Porto, J. A.; Justen, K. A.; Alves, N. D. C., Gresse Von Wangenheim, C., Borgatto, A. F.; Hauck, J. C. R. Codemaster Ui Design – App Inventor: Uma Rubrica De Avaliação Do Design De Interface De Aplicativos Android Desenvolvidos Com App Inventor. Proc. Of The Xvii Simpósio Brasileiro Sobre Fatores Humanos Em Sistemas Computacionais, Vitória, Brasil, 2019.

Ting Su, Guozhu Meng, Yuting Chen, KeWu,Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. 2017. Guided, Stochastic Model-based GUI Testing of Android Apps. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017). ACM, New York, NY, USA, 245–256.

Ultralytics. “YOLO11 NEW”. Ultralytics.com. Disponível em: <https://docs.ultralytics.com/models/yolo11/>. Acesso em: 5 dez. 2024.

Veronica Cateté, Erin Snider and Tiffany Barnes (2016). Developing a Rubric for a Creative CS Principles Lab. Proc. of the ACM Conference on Innovation and Technology in Computer Science Education, Arequipa, Peru.

W3C World Wide Web Consortium, “Web Content Accessibility Guidelines 2.0” (2008). Disponível em: <https://www.w3.org/TR/2008/REC-WCAG20-20081211/>. Acesso em: 12 abr. 2019.

Wangenheim, C. G. V. et al. CodeMaster – Automatic Assessment and Grading of App Inventor and Snap! Programs. Informatics In Education, [s.l.], v. 17, n. 1,

p.117-150, 20 abr. 2018. Vilnius University Press. http://dx.doi.org/10.15388/infedu.2018.08.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

X. Xiao, X. Wang, Z. Cao, H. Wang, and Peng Gao. 2019. IconIntent: automatic identification of sensitive UI widgets based on icon classification for Android apps. In Proc.of the 41st Int. Conf.on Software Engineering (ICSE '19). IEEE Press, Piscataway, NJ, USA, 257-268.

Zhao, Z., Zheng, P., Xu, S. Et Al. “Object Detection with Deep Learning: A Review”. arXiv (Cornell University), 2018.

Zou, Z. et al. “Object Detection in 20 Years: A Survey”, arXiv:1905.05055v2 [cs.CV], 2019.

APÊNDICE A – Artigo

Usando Machine Learning para Extração da Semântica de Design de Elementos de Interface de Usuário de Aplicativos do App Inventor

João Vitor Araujo Porto
Dep. de Informática e Estatística
Universidade Federal de Santa Catarina
Florianópolis SC Brasil
joao.porto@grad.ufsc.br

Maurício Floriano Galimberti
Dep. de Informática e Estatística
Universidade Federal de Santa Catarina
Florianópolis SC Brasil
m.f.galimberti@ufsc.br

Abstract. *Computational education in K-12 Education is crucial in the context of the technological transformations of the 21st century. Using tools like App Inventor, students learn algorithms and programming, as well as develop user interfaces (UIs) for applications. Evaluating the design of these interfaces is an important aspect of the teaching and learning process. This paper proposes a machine learning-based model for extracting the design semantics of UIs in Android applications created with App Inventor, with the goal of integrating it into the CodeMaster tool. The work covered the theoretical foundation, state-of-the-art analysis, proposition, development, and evaluation of the model. The proposed model contributes to improving the evaluation of UI design, advancing the CodeMaster tool and enhancing computational education in K-12 Education.*

Resumo. *O ensino de computação na Educação Básica é crucial no contexto das transformações tecnológicas do século XXI. Utilizando ferramentas como o App Inventor, os alunos aprendem algoritmos e programação, além de desenvolverem interfaces de usuário (IUs) para aplicativos. A avaliação do design dessas interfaces é um aspecto importante do processo de ensino-aprendizagem. Este trabalho propõe um modelo baseado em machine learning para extrair a semântica de design de IUs de aplicativos Android criados com o App Inventor, com o objetivo de integrá-lo à ferramenta CodeMaster. O trabalho abrangeu a fundamentação teórica, análise do estado da arte, proposição, desenvolvimento e avaliação do modelo. O modelo proposto contribui para aprimorar a avaliação do design de IUs, promovendo avanços na ferramenta CodeMaster e no ensino de computação na Educação Básica.*

CCS CONCEPTS

• Human-centered computing ~ Interaction design ~ Interaction design process and methods ~ User interface design • Computing methodologies ~ Machine learning • Social and professional topics ~ Professional topics ~ Computing education

PALAVRAS CHAVES

Semântica de Design, Interface de Usuário, Usabilidade; Machine Learning, Deep Learning, App Inventor, Educação Básica.

1. Introdução

No século XXI, é essencial que os alunos não apenas consumam, mas também criem e aperfeiçoem tecnologias, o que torna o ensino de conceitos e técnicas de computação fundamental na Educação Básica [CSTA 2017]. A computação pode ser ensinada por meio do desenvolvimento de aplicativos móveis, onde conceitos como abstração, recursão, iteração e análise de dados são aplicados durante a criação de aplicativos, além de aproveitar a familiaridade da população com smartphones [Grover e Pea 2013; Kepios Pte. Ltd. 2019]. O App Inventor, uma ferramenta de programação visual baseada em blocos, permite que qualquer pessoa, mesmo sem experiência em programação, desenvolva aplicativos Android [MIT 2019].

O desenvolvimento de aplicativos não se restringe à programação das funcionalidades, mas inclui o design das interfaces de usuário (IUs), que é crucial para garantir a usabilidade e a experiência do usuário. Ensinar design de interface visa maximizar a interação eficaz e satisfatória entre o usuário e o aplicativo, e isso tem sido cada vez mais incorporado no ensino de computação [Ferreira et al. 2019; Nielsen 1993]. Para isso, é importante que os aplicativos sejam desenvolvidos em conformidade com heurísticas e guias de estilo, como o Material Design e as diretrizes do W3C.

No entanto, fornecer feedback instrucional eficaz sobre o design de IUs é desafiador, especialmente na educação básica, onde a avaliação manual é trabalhosa e difícil de escalar. A escassez de professores especializados em computação e o fato de muitos professores de outras áreas introduzirem a computação de maneira interdisciplinar agravam a dificuldade em avaliar corretamente, além de a avaliação manual ser suscetível a erros como inconsistência e fadiga [Eseryel et al. 2013; Grover et al. 2014; Cateté, Snider e Barnes 2016; Zen, Iskandar e Linang 2011]. A avaliação automatizada se apresenta como uma solução viável, reduzindo a carga de trabalho dos professores e permitindo que eles se concentrem em outras atividades com os alunos [Ala-Mutka e Järvinen 2004].

Embora existam ferramentas como o CodeMaster para avaliar algoritmos e programação em aplicativos criados no App Inventor, elas não abordam adequadamente o design de IUs [Alves et al. 2019; Wangenheim et al. 2018]. A ferramenta CodeMaster - UI Design avalia o design com base em rubricas, mas ainda carece de informações semânticas sobre os elementos de IU, como o tamanho adequado da fonte para diferentes tipos de texto [Google Design 2019; Porto et al. 2018]. Este trabalho propõe a criação de um modelo automatizado de extração semântica de elementos de design de IUs, utilizando técnicas de machine learning, como redes neurais e clusterização, para tornar a avaliação mais completa e aplicável ao contexto educacional dos aplicativos criados com o App Inventor [Liu et al. 2018; Degaki et al. 2022].

2. Estado da arte

Para levantar o estado da arte e identificar modelos que utilizam machine learning para a detecção de elementos em interfaces de usuário (IUs) de aplicativos Android, foi realizado um mapeamento sistemático da literatura, com base no protocolo de pesquisa proposto por Petersen (2008), focando em artigos publicados entre 2009 e 2024. A pesquisa foi estruturada para responder a questões sobre os tipos de elementos detectados, os conjuntos de dados utilizados, os algoritmos de machine learning empregados, e a avaliação da qualidade dos resultados [Petersen 2008]. Os critérios de inclusão consideraram apenas estudos focados na

detecção de IUs por meio de técnicas de machine learning, com artigos excluídos caso não fornecessem informações suficientes ou não abordassem a temática central.

A busca foi conduzida em diversas bases de dados, incluindo ACM Digital Library, IEEE Xplore, Scopus, SpringerLink e Wiley, além de Google Scholar e Google para ampliar o escopo [Haddaway et al. 2015]. O processo envolveu a calibração de strings de busca para garantir a abrangência necessária. Após a análise preliminar dos artigos, apenas cinco foram selecionados como relevantes, com foco na detecção de elementos de IUs em aplicativos móveis [Petersen 2008]. Em 2024, uma nova busca foi realizada, resultando em um artigo adicional relevante. Os estudos encontrados geralmente usaram machine learning como um meio para a geração de código ou para automatizar a detecção de elementos visuais.

Quanto aos elementos detectados, a maioria dos estudos focou em IUs para Android, com exceção de um estudo que abordou também outras plataformas [Chen et al. 2019]. A detecção envolveu componentes típicos de hierarquias de IUs dos aplicativos Android, com modelos focados em grandes conjuntos de dados extraídos da Google Play Store. Esses conjuntos de dados foram, em sua maioria, gerados de forma automatizada, com algumas exceções que utilizaram rotulagem manual [Liu et al. 2018; Degaki et al. 2022]. A maioria dos modelos usou redes neurais convolucionais (CNNs), técnicas de aprendizado supervisionado que se mostraram eficazes para a predição de imagens [Moran et al. 2018; Liu et al. 2018].

Em termos de performance, os modelos apresentaram bons resultados, superando métodos alternativos, como máquinas de vetores de suporte (SVM) e regressão [Moran et al. 2018; Liu et al. 2018]. No entanto, nenhum modelo conseguiu atingir uma precisão superior a 95%, sugerindo que ainda há espaço para melhorias [Moran et al. 2018]. Embora os avanços sejam significativos, a área de detecção de elementos de IUs usando machine learning permanece em estágio inicial, com poucos trabalhos dedicados exclusivamente a essa temática. Isso indica a necessidade de novos modelos que possam melhorar os resultados de desempenho e expandir o uso para diferentes plataformas de aplicativos móveis [Chen et al. 2019; Liu et al. 2018].

3. Desenvolvimento do modelo

O modelo desenvolvido neste trabalho, denominado "TifSX" (acrônimo para "TifSX is for Semantics eXtraction"), foi construído para identificar elementos de interfaces de usuário (IUs) e extrair sua semântica, visando sua integração com a ferramenta CodeMaster. O desenvolvimento seguiu as etapas descritas na Figura 9, que incluem: análise de requisitos, seleção do modelo base, criação do conjunto de dados, treinamento, inferência, exportação do modelo e teste.

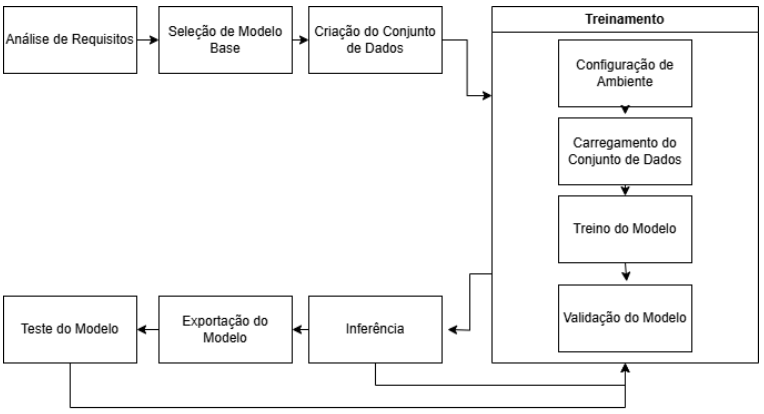


Figura 1: Fluxo de desenvolvimento do modelo TifSX

Análise de Requisitos: O modelo foi projetado para identificar e extrair a semântica de elementos de IUs. Os requisitos funcionais e não funcionais foram definidos para garantir que o modelo atendesse às necessidades de detecção e classificação em tempo real.

Seleção de Modelo Base: Várias abordagens foram consideradas para a detecção de objetos em IUs, incluindo métodos clássicos (Detecção de Borda, Correspondência de Template) e abordagens de machine learning (CNNs de Dois Passos, CNNs de Um Passo e Híbridas) [Degaki et al., 2022]. Optou-se pelo uso de CNNs, especificamente o modelo YOLO, que é eficaz para detecção em tempo real devido à sua alta velocidade e precisão (Redmon & Farhadi, 2018). A escolha do YOLO foi baseada no equilíbrio entre precisão e velocidade, crucial para a integração com o CodeMaster [Wangenheim et al. 2018].

Criação do Conjunto de Dados: Utilizou-se o conjunto de dados RICO, que contém 66.261 IUs de 9.772 aplicativos Android (Deka et al., 2017). Este conjunto inclui imagens, hierarquias de elementos e anotações semânticas, que foram convertidas para o formato esperado pela CNN YOLO [Porto, 2024].

Treinamento: O treinamento foi realizado utilizando a versão 11 da implementação do YOLO da Ultralytics [Ultralytics 2024], com técnicas de transfer learning para aproveitar modelos pré-treinados [Goodfellow et al. 2016]. A versão "small" do modelo foi escolhida por equilibrar eficiência e tempo de resposta. O treinamento ocorreu no ambiente Google Colab, usando a GPU NVIDIA A100, ideal para processamento intensivo de deep learning (Nvidia, 2024).

Inferência: O desempenho do modelo foi avaliado utilizando métricas como Intersection over Union (IoU), precision, recall, average precision e Mean Average Precision (mAP) [Zou et al. 2019]. O modelo alcançou um mAP@0.5 de 81,56% e um mAP@0.95 de 67,41%, com F1 score de 76,17%, evidenciando boa precisão na detecção dos elementos da IU.

Exportação: O modelo foi exportado [Porto 2024] para os formatos ONNX [Onnx 2024] e TensorRT [Nvidia, 2024], ampliando sua aplicabilidade em diferentes dispositivos e ferramentas, como o CodeMaster, garantindo eficiência e alto desempenho.

Esse processo resultou em um modelo robusto e eficiente para a extração semântica de elementos em IUs, adequado para sua futura integração com a ferramenta CodeMaster.

4. Resultados

A avaliação do modelo "TifSX" foi realizada para analisar sua correspondência na detecção de elementos de IUs no App Inventor e o tempo de resposta para essas detecções.

Utilizando práticas recomendadas para análise de modelos de detecção com redes neurais convolucionais [Zhao et al. 2018], o modelo foi testado com um conjunto de dados exclusivo, composto por 100 imagens de IUs rotuladas manualmente, garantindo um teste independente do conjunto de treinamento e evitando overfitting [Goodfellow et al. 2016]. Os resultados obtidos mostraram que o modelo alcançou um mAP@0.5 de 72,40%, o que indica um bom desempenho, mas o mAP@0.95 de 53,84% sugere dificuldades em atender aos critérios mais rigorosos de localização e classificação [Lin et al. 2014]. O F1 score de 71,16% também indicou um equilíbrio razoável entre precisão e recall, mas com margem para melhorias, especialmente no ajuste fino do modelo, como ajuste de hiperparâmetros e uso de transfer learning com modelos mais robustos [Robinson 2019].

Em termos de tempo de resposta, o modelo apresentou um tempo médio de inferência de 63 ms por imagem, o que é adequado para aplicações em tempo real, como as exigidas pela ferramenta CodeMaster, sem comprometer a usabilidade.

Embora o modelo tenha demonstrado desempenho satisfatório, há espaço para melhorar tanto as métricas de detecção quanto a quantidade de semântica extraída, através da exploração de mais dados de anotações, como as do conjunto RICO, o que, no entanto, exigiria maior capacidade computacional. Comparado a modelos anteriores focados na detecção de IUs [Degaki et al. 2022], o TifSX apresentou um desempenho superior, consolidando o uso de modelos baseados em YOLO para essa tarefa. O modelo, ao ser integrado à ferramenta CodeMaster, mostra-se viável para a extração de semântica de design de elementos de IU, oferecendo uma solução moderna e eficiente que pode ser aplicada mesmo em ambientes com hardware limitado, como aqueles encontrados na Educação Básica, devido ao seu baixo tempo de resposta.

7. Conclusão

O modelo TifSX, desenvolvido para a detecção de objetos em interfaces de usuário de aplicativos Android criados no App Inventor, demonstrou um desempenho satisfatório, com métricas de avaliação que atendem aos padrões de arquiteturas de detecção do tipo YOLO, como mAP baseado em Intersection over Union. Os resultados indicam que o modelo é eficaz na identificação e classificação de componentes de IUs, superando modelos anteriores na literatura para tarefas similares, o que confirma sua relevância no contexto de design de interfaces gráficas. A possibilidade de integração do TifSX com a ferramenta CodeMaster tem o potencial de ampliar suas capacidades, permitindo feedback automatizado sobre o design de aplicativos criados por estudantes, o que pode impulsionar o processo de ensino-aprendizagem e promover melhorias contínuas no design de interfaces no contexto educacional. Embora o modelo tenha mostrado um bom desempenho, existem margens para melhorias, especialmente nas métricas de precisão. Trabalhos futuros incluem a integração do modelo ao CodeMaster e o ajuste fino do modelo com a exploração de novos hiperparâmetros e conjuntos de dados mais amplos, visando aprimorar sua performance.

REFERÊNCIAS

ALA-MUTKA, K.; JÄRVINEN, H.-M. (2004) “Assessment process for programming assignments”, In: Proc. of IEEE Int. Conference on Advanced Learning Technologies, Joensuu, Finland, p. 13-17.

ALVES, N. D. C.; WANGENHEIM, C. G. V.; HAUCK, J. C. R. (2019) “Approaches to Assess Computational Thinking Competences Based on Code Analysis in K-12 Education: A Systematic Mapping Study”, Informatics in Education, in press.

CATETÉ, V.; SNIDER, E.; BARNES, T. (2016) “Developing a Rubric for a Creative CS Principles Lab”, In: Proc. of the ACM Conference on Innovation and Technology in Computer Science Education, Arequipa, Peru, p. 92-97.

CSTA (2017) “Standards - CSTA K-12 Computer Science Standards”, Disponível em: https://www.csteachers.org/page/standards. Acesso em: 12 abr. 2019.

CHEN, S.; FAN, L.; SU, T.; MA, L.; LIU, Y.; XU, L. (2019) “Automated cross-platform GUI code generation for mobile apps”, In: Proc. of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER, p. 233-243.

ESERYEL, D.; IFENTHALER, D.; GE, X. (2013) “Validation study of a method for assessing complex ill-structured problem solving by using causal representations”, Educational Technology Research, v. 61, p. 443–463.

DEGAKI, R. H.; COLONNA, J. G.; LOPEZ, Y. et al. (2022) “Real Time Detection of Mobile Graphical User Interface Elements Using Convolutional Neural Networks”.

FERREIRA, M. N. F.; WANGENHEIM, C. G. V.; MISSFELDT FILHO, R.; PINHEIRO, F. D. C.; HAUCK, J. C. R. (2019) “Learning user interface design and development of mobile applications in middle school”, ACM Interactions, in press.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. (2016) Deep Learning, MIT Press.

GOOGLE DESIGN (2019) “Material Design”, Disponível em: https://material.io/design/. Acesso em: 12 abr. 2019.

GROVER, S.; PEA, R. (2013) “Computational Thinking in K–12: A Review of the State of the Field”, Educational Researcher, v. 42, n. 1, p. 38–43, doi:10.3102/0013189x12463051.

GROVER, S.; COOPER, S.; PEA, R. (2014) “Assessing Computational Learning in K-12”, In: Proc. of the Conference on Innovation & Technology in Computer Science Education, Uppsala, Sweden, p. 44-49.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. (2008) “Systematic Mapping Studies in Software Engineering”, In: Proc. of the 12th International Conference on Evaluation and Assessment in Software Engineering, Bari, Italy, p. 68-77.

PORTO, J. V. A.; BARBOSA, H.; WANGENHEIM, C. G. V. (2018) “Proposta de um Checklist de Avaliação de Usabilidade de Aplicativos Android no Contexto Educacional”, In: Computer on the Beach, Florianópolis, p. 112-119.

PORTO, J. V. A. (2023) TifSX, Disponível em: https://github.com/joaovaporto/TifSX. Acesso em: 23 dez. 2024.

KEPIOS PTE. LTD. (2019) “Digital 2019: Brazil – DataReportal – Global Digital Insights”, Disponível em: https://datareportal.com/reports/digital-2019-brazil. Acesso em: 12 abr. 2019.

LIU, T. F. et al. (2018) “Learning Design Semantics for Mobile Apps”, In: The 31st Annual ACM Symposium on User Interface Software and Technology - UIST '18, p. 569-579, ACM Press, http://dx.doi.org/10.1145/3242587.3242650.

MIT (2019) “MIT App Inventor News & Events | Explore MIT App Inventor”, Disponível em: http://appinventor.mit.edu/. Acesso em: 12 abr. 2019.

MORAN, K.; BERNAL-CÁRDENAS, C.; CURCIO, M.; BONETT, R.; POSHYVANYK, D. (2018) “Machine learning-based prototyping of graphical user interfaces for mobile apps”, IEEE Transactions on Software Engineering, v. 44, n. 11, p. 1103-1118.

NIELSEN, J. (1993) Usability Engineering, Morgan Kaufmann.

NVIDIA (2024) “NVIDIA A100 GPUs Power the Modern Data Center”, Disponível em: https://www.nvidia.com/en-us/data-center/a100/. Acesso em: 5 dez. 2024.

ONNX (2024) “Introduction to ONNX - ONNX 1.17.0 documentation”, Disponível em: https://onnx.ai/onnx/intro/. Acesso em: 5 dez. 2024.

ROBINSON, A. (2019) “Sketch2code: Generating a website from a paper mockup”, Disponível em: https://arxiv.org/abs/1905.13750. Acesso em: nov. 2019.

ULTRALYTICS (2024) “YOLO11 NEW”, Disponível em: https://docs.ultralytics.com/models/yolo11/. Acesso em: 5 dez. 2024.

WANGENHEIM, C. G. V. et al. (2018) “CodeMaster – Automatic Assessment and Grading of App Inventor and Snap! Programs”, Informatics in Education, v. 17, n. 1, p. 117-150, Vilnius University Press, http://dx.doi.org/10.15388/infedu.2018.08.

ZHAO, Z.; ZHENG, P.; XU, S. et al. (2018) “Object Detection with Deep Learning: A Review”, arXiv (Cornell University).

ZEN, K.; AWANG ISKANDAR, D. N. F.; LINANG, O. (2011) “Using Latent Semantic Analysis for Automated Grading Programming Assignments”, In: Proc. of the Int. Conference on Semantic Technology and Information Retrieval, Kuala Lumpur, Malaysia, p. 1-7.