UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CIÊNCIAS DA COMPUTAÇÃO

KLAUS DE FREITAS DORNSBACH

Desenvolvimento de um Modelo para Avaliação da Originalidade do Layout do Design de Interface de Aplicativos Android utilizando *Deep Learning*

FLORIANÓPOLIS
2022

KLAUS DE FREITAS DORNSBACH

Desenvolvimento de um Modelo para Avaliação da Originalidade do Design de Interface de Aplicativos Android utilizando *Deep Learning*

Trabalho de Conclusão de Curso de Graduação em Ciências da Computação, do Departamento de Informática e Estatística, do Centro Tecnológico da Universidade Federal de Santa Catarina, requisito parcial à obtenção do título de Bacharel em Ciências da Computação.

Orientadora: Prof.ª Dr.ª rer. nat. Christiane Gresse von Wangenheim, PMP

FLORIANÓPOLIS

2022

Resumo

Com o avanço da tecnologia no século XXI, novas habilidades se fazem necessárias na vida cotidiana e no mercado de trabalho. Dentre essas habilidades, destaca-se a criatividade, que é a capacidade de criar algo novo e útil. Uma das formas de estimular a criatividade é por meio da computação na Educação Básica, ensinando o desenvolvimento de apps utilizando o App Inventor, abordando não só a programação em si, mas também o design de interface. Por isso, como parte do processo de ensino-aprendizagem torna-se importante, também, a avaliação da originalidade dos *layouts* dos aplicativos criados pelos alunos. Nesse contexto, o objetivo deste trabalho é o desenvolvimento de um modelo automatizado, utilizando técnicas de *deep learning* para avaliar a originalidade da interface de aplicativos desenvolvidos com o App Inventor, para ser utilizado no ensino de computação na Educação Básica. A pesquisa inclui fundamentação teórica acerca de design de layout e criatividade, assim como técnicas de machine learning e suas métricas. Espera-se contribuir para melhorar então a aprendizagem da habilidade de criatividade fornecendo *feedback* aos alunos e professores no processo de aprendizagem.

Palavras chave: Originalidade, Criatividade, Estética Visual, *App Inventor*, Design de Interface de Usuário, Educação Básica, *Deep Learning*.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. Introdução

A criatividade é uma habilidade essencial no século XXI (CAVALLO, 2016), logo o desenvolvimento dessa habilidade deve fazer parte da formação do indivíduo desde cedo. Entre outras disciplinas ela também pode ser desenvolvida pelo ensino de computação (ALVES et al., 2020). Uma forma de ensinar computação na educação básica pode ser por meio de desenvolvimento de apps com App Inventor (appinventor.mit.edu), incluindo o projeto de design de interface como também a programação do parte funcional do app (FERREIRA, 2020).

Design de interface de usuário é o processo pelo qual designers constroem interfaces, ou portas para interação, para usuários de software ou dispositivos computadorizados. Um design de interface de aplicativos engloba o *layout,* que é a forma em que elementos ficam dispostos numa página.

No contexto do ensino de computação, uma das estratégias mais usadas é a metodologia ativa usando o ciclo “Use-Modifique-Crie” (UMC) (ALVES, 2020). Seguindo esse ciclo o aluno aprende conceitos importantes analisando de um programa pronto, para que posteriormente possa criar um artefato próprio, demonstrando também sua criatividade por meio de um *layout* de Interface de usuário (UI). Nessas circunstâncias, é importante uma avaliação e *feedback* acerca da aprendizagem da criatividade, ajudando o estudante no seu processo de aprendizagem. Entretanto, avaliações manuais feitas por um ser humano estão sujeitas a serem subjetivas e trabalhosas, uma alternativa para mitigar o risco de avaliações injustas e poupar tempo dos professores de computação é a automatização desse processo (ALVES, 2020).

Para criar um avaliador automático é necessário uma definição de criatividade do produto. Mesmo sendo um conceito complexo, existem algumas

definições que revolvem ao redor da ideia de criatividade incluir utilidade, adequação e a originalidade (BESEMER e TREFFINGER, 1981). Consequentemente é necessário que uma avaliação de criatividade contemple a dimensão da originalidade. Porém a maioria das abordagens de avaliação automática existentes no contexto do ensino de computação avaliam originalidade por meio da divergência entre códigos fonte, não abordando características visuais como design de interface, nem características funcionais (ALVES, 2020).

Assim, a proposta deste trabalho de conclusão de curso é criar um modelo que automaticamente avalie o design de interface criado pelo estudante em relação a originalidade do *layout* de UI no contexto de aplicativos criados no App Inventor. Já existem abordagens, inclusive pesquisas realizadas por pesquisadores da iniciativa Computação na Escola, em particular Souza (2022) e Kreuch (2022) que utilizam abordagens supervisionadas para identificar componentes do layout de um aplicativo e medir similaridade com base nisso, calculando uma nota de originalidade. Kreuch (2022) utilizou um subconjunto de aproximadamente 4% do conjunto de dados para o treino do módulo que realiza reconhecimento de componentes de interface, abaixo da média de 70% nos estudos avaliados na pesquisa bibliográfica,

Diferente destas abordagens usando aprendizagem supervisionada, o presente trabalho visa utilizar métodos não supervisionados para aumentar o tamanho do conjunto de dados de treino por meio de treinamento não supervisionado.

Portanto, foi realizada uma pesquisa acerca da originalidade no design de *layouts* e visa-se a adoção de técnicas de *deep learning*, treinando um modelo para avaliar automaticamente a originalidade de *layouts,* procurando utilizar métodos diferentes dos já explorados, de maneira a compará-los e melhorar a desempenho.

1.2. Objetivos

Objetivo geral

O objetivo geral do trabalho é utilizar técnicas alternativas de *deep learning* para criar um modelo capaz de atribuir um nível de originalidade a um aplicativo,

baseado em quão diferente suas telas são comparadas ao universo de referência. Este modelo tem como objetivo ser utilizado no contexto educacional, mais especificamente para automatizar a avaliação de aplicativos feitos utilizando app inventor.

Objetivos Específicos

O1. Analisar a fundamentação teórica sobre originalidade, design de interfaces de usuário e similaridade de *layouts*/estrutura de imagens e *deep learning*.

O2. Analisar o estado da arte em relação a análise automática da originalidade do *layout* de design de interfaces de apps.

O3. Selecionar e aplicar técnicas de avaliação de similaridade, aplicando-as e avaliando seu desempenho.

Premissas e restrições

O trabalho é realizado de acordo com o regulamento vigente do Departamento de Informática e Estatística (INE – UFSC) em relação aos Trabalhos de Conclusão de Curso. O modelo é desenvolvido para uso somente no contexto de aplicativos Android, mais especificamente feitos com o App Inventor.

1.3. Metodologia de pesquisa

A metodologia de pesquisa utilizada neste trabalho é dividida nas seguintes etapas.

Etapa 1 – Fundamentação teórica

Estudando, analisando e sintetizando os conceitos principais e a teoria referente aos temas a serem abordados neste trabalho é apresentado a fundamentação teórica utilizando a metodologia de revisão narrativa (Cordeiro et al., 2007). Nesta etapa são realizadas as seguintes atividades:

A1.1 Análise teórica sobre originalidade no contexto da criatividade

A1.2 Análise teórica sobre design de interface de apps

A1.3 Análise teórica sobre computação de similaridade utilizando *deep learning*

Etapa 2 – Estado da arte

Nesta etapa é realizado um mapeamento sistemático da literatura seguindo o processo proposto por Petersen et al. (2015) para identificar e analisar modelos de análise automatizado da originalidade de design de interfaces de usuário de apps atualmente sendo utilizados. Esta etapa é dividida nas seguintes atividades:

A2.1 – Definição do protocolo da revisão

A2.2 – Execução da busca e seleção de artigos relevantes

A2.3 – Extração e análise de informações relevantes

Etapa 3 – Desenvolvimento

Nesta etapa são selecionadas técnicas apropriadas de *deep learning* para análise da originalidade do *layout* de design de interfaces de apps criados no App Inventor. A partir da seleção foram implementadas essas técnicas e analisados e comparados os seus desempenhos seguindo um processo de desenvolvimento de redes neurais/*deep learning* (AMERSHI et al, 2019). Esta etapa é dividida nas seguintes atividades:

A3.1 – Análise de requisitos

A3.2 – Preparação de conjunto de dados de universo de referência e um conjunto de teste

A3.4 – Seleção de técnicas de *deep learning*

A3.5 – Implementação das técnicas selecionadas

A3.6 – Avaliação e comparação do desempenho

2. Fundamentação teórica

2.1 Originalidade no contexto da criatividade

A criatividade é uma habilidade essencial no século XXI (CAVALLO et al., 2016), logo o desenvolvimento dessa habilidade deve fazer parte da formação do indivíduo desde cedo. Entre outras competências ela também pode ser desenvolvida pelo ensino de computação (ALVES et al., 2020). Uma forma de ensinar computação na educação básica pode ser por meio de desenvolvimento de apps com App Inventor (GROVER, BASU e SCHANK, 2018). Esse ensino por meio de desenvolvimento de apps tipicamente ensina conceitos de algoritmos e programação, mas pode também incluir o design de interface (ALVES et al., 2020).

Nesse contexto educacional, é importante uma avaliação e *feedback* acerca da aprendizagem da criatividade, ajudando o estudante no seu processo de aprendizagem (ALVES et al., 2020). A tarefa de avaliar a criatividade é complexa (ALVES et al., 2020). Para operacionalizar a avaliação da criatividade, essa competência pode ser subdividida. Uma divisão possível é os 4 P's da criatividade (BESEMER e TREFFINGER, 1981): produto, pessoa, ambiente (*press*) e processo (RHODES, 1961). Cada um desses âmbitos possui parâmetros de criatividade diferentes, por exemplo, o que faz um ambiente ser criativo é diferente do que faz um produto. No contexto deste trabalho, são considerados parâmetros avaliativos voltados ao produto, visto que é o que mais se encaixa voltado ao objetivo de avaliação por desempenho dentro do contexto de ensino de computação resultando na criação de aplicativos criativos.

Um produto pode ser definido como sendo um objeto físico, um artefato computacional, como p.ex. um app, que não está unicamente ligado com a vida de um indivíduo (BROGDEN e SPRECHER, 1964). A definição de criatividade focada no produto geralmente inclui características como novidade (quão incomum ou original o produto é), adequação (se o produto serve ao seu propósito e este propósito é útil) e condensação (quão atrativo o produto é) (BESEMER e TREFFINGER, 1981). Entre essas características, o presente trabalho foca especificamente na característica de novidade. A novidade de um aplicativo modelo pode se relacionar a diversos planos (Figura 1) especificamente na originalidade do design de interface.

Plano		Inclui
Objetivo		**Objetivos do sistema de software** Grau de diferença em relação ao objetivo do aplicativo.
Escopo		**Especificações funcionais** Grau de diferença em relação às funcionalidades dos aplicativos.
		Requisitos de conteúdo Grau de diferença em relação ao conteúdo dos aplicativos.
Design de UI	**Estrutura**	**Design de interação** Grau de diferenças com relação à entrada/saída de aplicativos (sensores, mídia, conectividade, etc).
		Arquitetura de informação Grau de diferenças em relação às informações apresentadas pelos aplicativos.
	Esqueleto	**Design de interface** Grau de diferenças em relação aos componentes da GUI, posicionamento e novas interfaces.
		Design de navegação Grau de diferenças em relação ao fluxo de navegação.
		Design de informação Grau de diferenças em relação à maneira como as informações são apresentadas.
	Aparência	**Design visual** Grau de diferenças com relação às cores, tipografia e imagens e ícones.
Código		**Código fonte** Grau de diferenças em relação ao JSON do projeto (representado como um vetor de recursos)

Figura 1. Dimensões de novidade em um aplicativo (Fonte: ALVES, 2020)

A novidade se refere a quantidade e a que extensão novos conceitos, técnicas e materiais são incluídos num produto (BESEMER e TREFFINGER, 1981). A originalidade se deriva a partir deste conceito e é um dos mais citados parâmetros para avaliação de criatividade, podendo ser definida como algo raro (GUILFORD, 1950). Mas algo só pode ser definido como novo tendo-se uma referência do que já foi criado. Por consequência, para analisarmos a originalidade de um *layout* de um design de interface de app é necessário considerar um universo de referência (um espaço amostral) de outros *layouts* já desenvolvidos em contexto semelhantes e formas de relacionar o grau de similaridade entre estes e aquilo que é novo.

2.2 Design de interface de apps

O design é o esforço consciente ou intuitivo de impor ordem e significado (PAPANEK, 1971) e busca conciliar estética e funcionalidade (SCHLATTER e LEVINSON, 2013). Nesse contexto, faz parte do design de um aplicativo o *layout* de uma interface, ou seja, a definição dos elementos e sua composição numa tela. Idealmente se busca compor uma hierarquia que comunique visualmente

funcionalidades ao usuário com clareza, para uma boa experiência de usuário. A confecção de um *layout* deve prezar por uma boa comunicação pois *layouts* criados mantendo o receptor em mente são em geral mais atraentes e de melhor inteligibilidade para estes (SCHLATTER e LEVINSON, 2013).

Saber o tamanho da tela e o seu comportamento é de importância na criação do *layout*, pois a tela fornece a estrutura onde o conteúdo será apresentado (SCHLATTER e LEVINSON, 2013). O menor elemento de uma tela é o pixel, e cada tela possui uma relação altura x largura definida em pixels chamada de resolução, entretanto as dimensões reais do pixel variam de aparelho para aparelho. Por isso é utilizado por padrão na indústria o dip (*device independent pixels*) uma unidade de comprimento que depende da densidade de pixels na tela e é definida por cada fabricante e mapeada por debaixo dos panos na execução da aplicação.

Outro aspecto importante do *layout* é o posicionamento de elementos. Sempre que há mais de um elemento em tela, relações visuais são criadas entre esses, criar um *layout* envolve posicionar elementos para formar relações úteis e perceptíveis (SCHLATTER e LEVINSON, 2013), o alinhamento guia o usuário na medida que forma relações entre elementos da interface.

Dentro dos conceitos necessários no design de *layouts* está a ideia de espaço em branco. É o espaço negativo, o “nada” ao redor do conteúdo (SCHLATTER e LEVINSON, 2013). Apesar de ser chamado de espaço branco, sua cor não é necessariamente branca, é um espaço que contém somente alguma cor menos saturada utilizado para descansar o receptor, limitando a quantidade de informação recebida. O espaço em branco é importante para legibilidade e para criar uma hierarquia e relação dos elementos, levando a atenção do usuário as funcionalidades mais importantes.

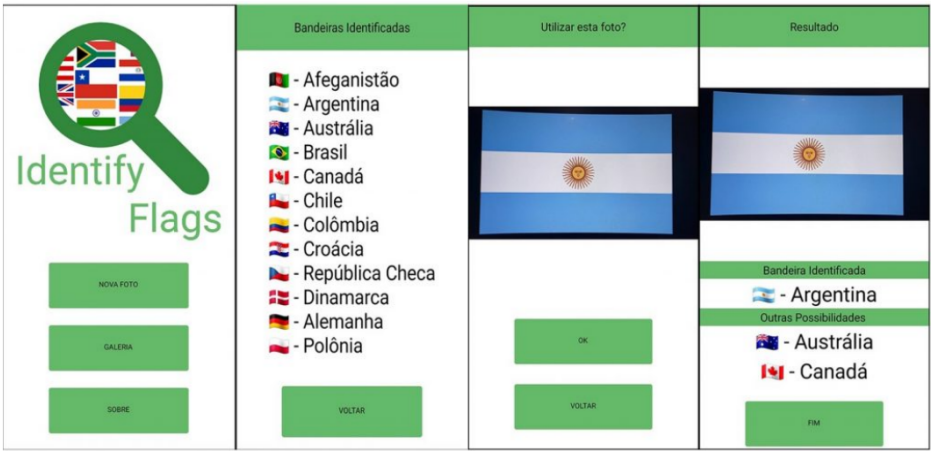


Figura 2. O posicionamento vertical dos elementos de interface dos botões cria um agrupamento e uma hierarquia de elementos (Fonte: BAULÉ, 2020)

Margens são um tipo de espaço em branco entre as bordas de um contêiner e os elementos dentro deste, estando presente sempre que há um elemento dentro de outro (SCHLATTER e LEVINSON, 2013). Esse espaço é utilizado para criar designs mais confortáveis (Figura 3).

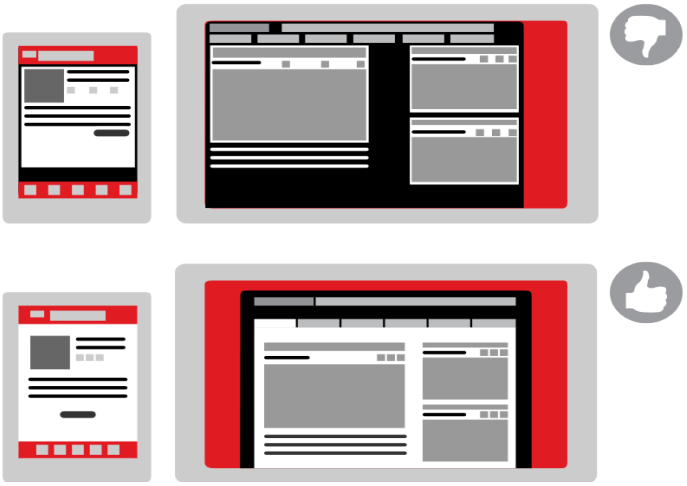


Figura 3. Exemplificação de uso inadequado e adequado de margens. (Fonte: SCHLATTER e LEVINSON, 2013)

Escala, ou tamanho relativo dos elementos, está diretamente relacionado ao peso visual e a interpretação de importância. Quando elementos são do mesmo tamanho, eles podem parecer relacionados. Quando se utiliza escala para indicar importância e definir hierarquia, diferenças de tamanho precisam ser claras (SCHLATTER e LEVINSON, 2013).

Alinhamento é outro meio de organizar o *layout*, expondo relacionamentos entre elementos. Alinhamento é utilizado na tipografia, geralmente definindo alinhamentos com base em uma linha de base posicionada à esquerda, centralizada ou à direita (Figura 4).

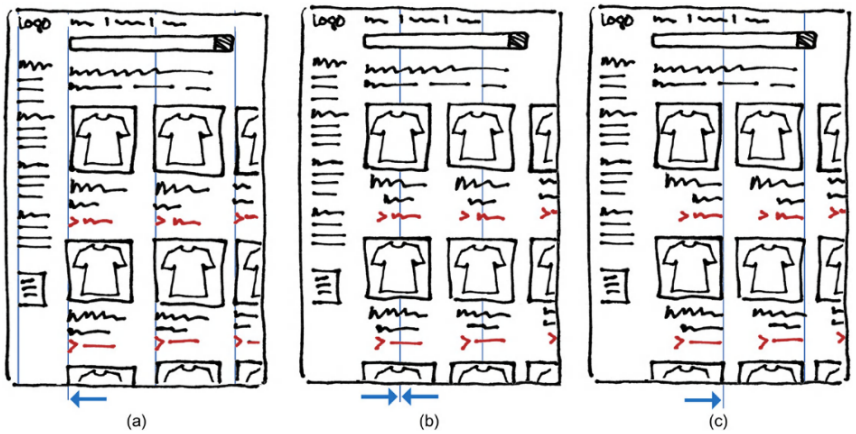


Figura 4. Exemplos de alinhamento de tipografia. (Fonte: SCHLATTER e LEVINSON, 2013).

2.3 Design de interfaces no App Inventor

O App Inventor (appinventor.mit.edu) permite a criação de aplicativos por meio de interface de usuário (PATTON et al., 2019). A parte gráfica de interface de usuário é definida arrastando e posicionando blocos com elementos visuais, e na parte de programação destes elementos, são disponibilizados blocos que se encaixam e formam sequências de comandos. Existe uma distinção entre um ambiente de design de interface (Figura 5), e de programação (Figura 6) em que sequências de instruções são definidas. Devido à sua simplicidade, ambientes de programação baseado em bloco como o App Inventor são utilizados no ensino de computação (MIT, 2021)

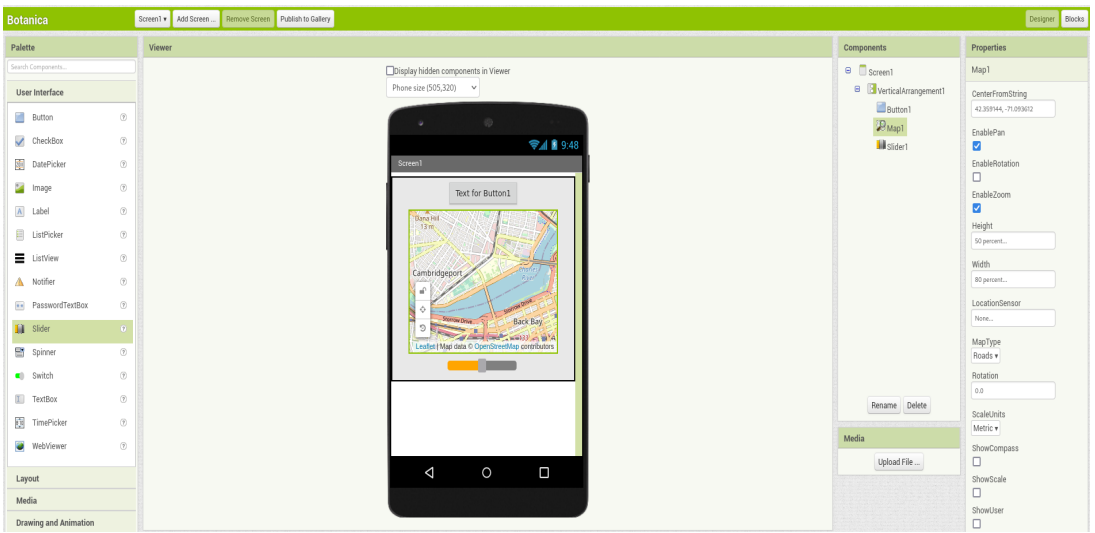


Figura 5. Editor de design de interface. (Fonte: próprio autor).

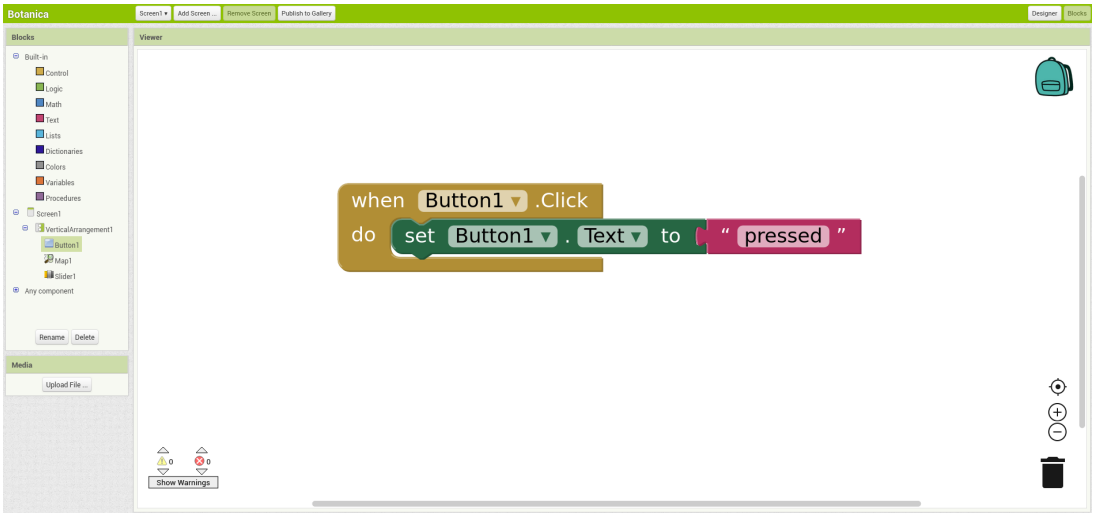


Figura 6. Editor de código em blocos do app inventor. (Fonte: próprio autor).

O App Inventor possui diversos elementos de *layout* e formas de posicioná-los. A Tabela 1 apresenta os elementos de interface disponíveis no App Inventor. Elementos sem visualização como sons e sensores foram excluídos.

Tabela 1. Elementos de interface de usuário no App Inventor

Elemento	**Breve descrição**
Button	Botão que detecta cliques
CheckBox	Cria um evento quando o usuário clica
DatePicker	Lança uma caixa de diálogo para escolher data
Image	Exibe imagem
Label	Exibe texto
ListPicker	Exibe lista de textos para usuário escolher dentre opções
ListView	Exibe lista de diversos componentes como imagem e texto
PasswordTextBox	Caixa de texto com exibição dos caracteres protegida
Slider	Barra arrastável que permite entradas que vão de um mínimo a um máximo
Spinner	Exibe pop up com lista de elementos
Switch	Botão que armazena estado e cria eventos nas transições
TextBox	Permite *entrada* de texto do usuário
TimePicker	Permite o usuário escolher um horário por meio de um pop up
WebViewer	Permite visualização de páginas web
Map	Contêiner bidimensional que renderiza um mapa no plano de fundo
ImagePicker	Permite usuário escolher imagem da galeria
VideoPlayer	Componente multimídia que permite controlar vibração e exibir vídeos
Canvas	Painel retangular que permite usuário desenhar e mover sprites
Chart	Elemento que possibilita visualização de dados em gráficos

Entre os elementos de interface do usuário, existem alguns com aparência idêntica (Figura 7). Por exemplo, elementos do tipo picker que abrem um pop up de interação com o usuário são indistinguíveis de um botão antes de serem pressionados. No contexto do presente trabalho que propõe um modelo que necessita diferenciar elementos de interface, componentes com visualização idêntica são um problema. A Tabela 2 mostra os elementos que tem uma visualização igual.

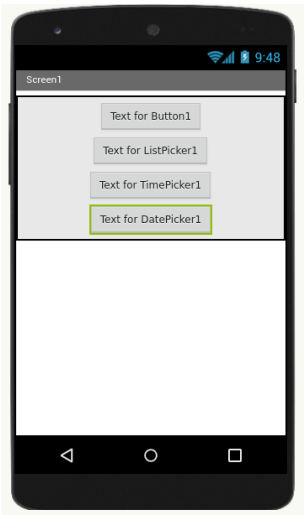


Figura 7. Exemplos de elementos de interface com aparência idêntica no App Inventor (Fonte: próprio autor)

Tabela 2. Elementos do app inventor com visualização idêntica.

Elemento base	**Elemento**
Button	DatePicker
	ListPicker
	TimePicker
	ImagePicker
	ContactPicker
	PhoneNumberPicker
TextBox	EmailPicker
Image	Canvas

Outro problema são os elementos que não possuem um padrão visual definido. Elementos como *web viewer* tem sua aparência atrelada a um link, não sendo possível detectar algum padrão para detectar este tipo de elemento (Figura 8).

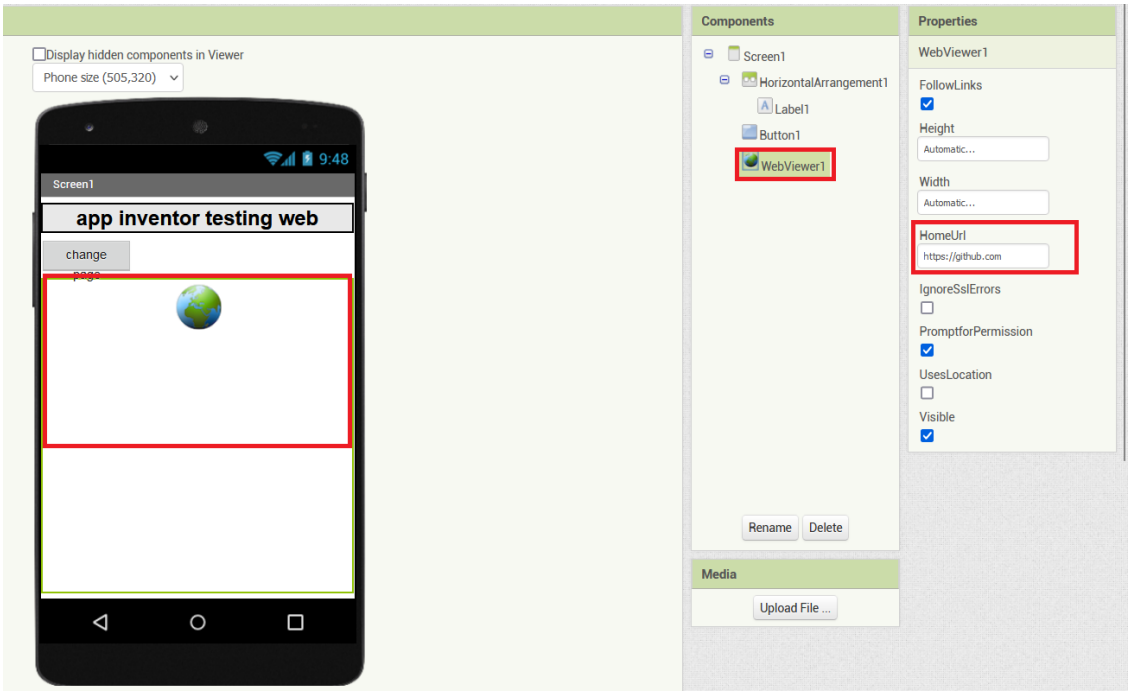


Figura 8. Projeto com web viewer. (Fonte: próprio autor)

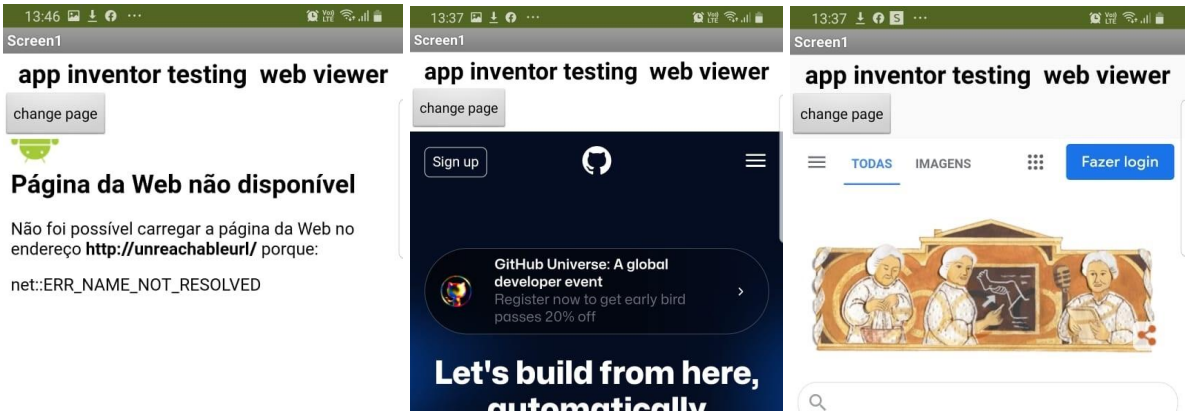


Figura 9. Exemplos de aplicativos App Inventor com diferentes páginas linkadas no *web viewer*. (Fonte: próprio autor)

No contexto de alinhamento, o App Inventor possui diversas formas de organizar os elementos de um *layout*. O conceito de encapsulamento é importante para a organização de um *layout*, é quando um elemento envolve outro dentro de

seus limites na interface. Há a possibilidade de editar o alinhamento por meio de um “arranjo”, elemento de interface que encapsula outros elementos e define automaticamente o alinhamento de todos.

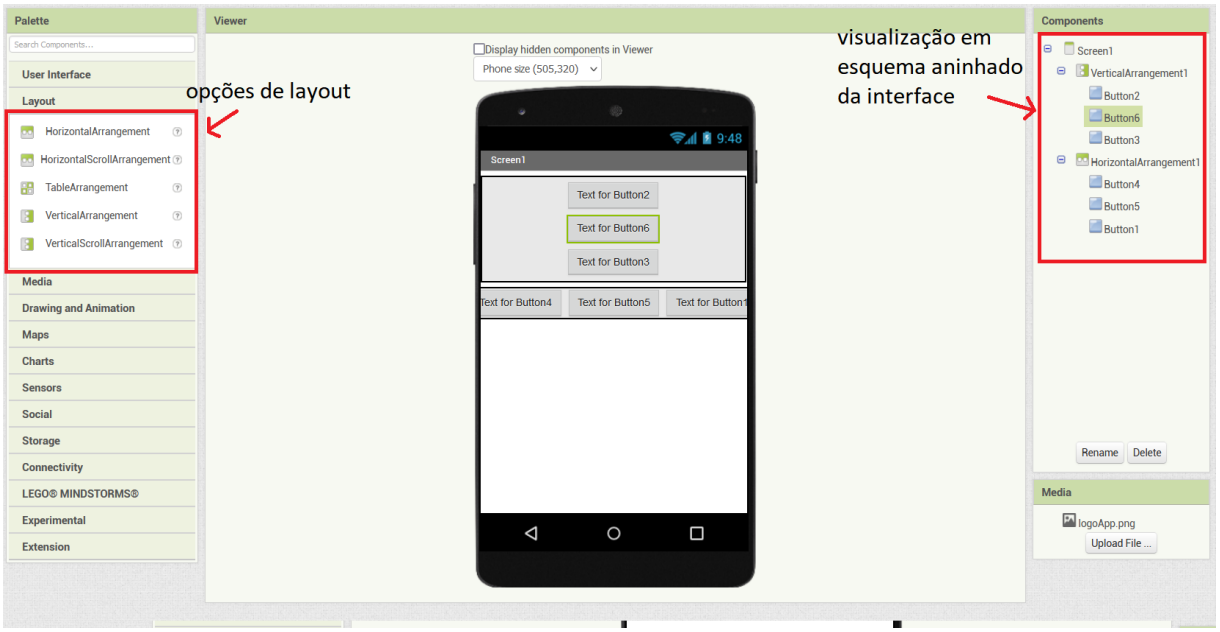


Figura 10. Elementos de alinhamento, sua visualização na interface e no esquema. (Fonte: próprio autor).

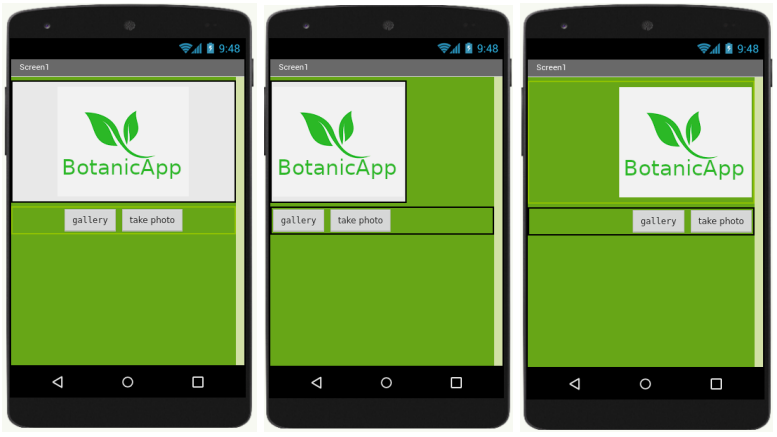


Figura 11. Alinhamento centralizado, à esquerda e à direita do *layout* de um app desenvolvido com App Inventor. (Fonte: próprio autor).

Além do posicionamento, pode ser definida a altura e largura dos elementos individualmente em suas propriedades, existe a possibilidade de definir largura e altura por pixels, porcentagem, automático para ocupar o espaço necessário para exibir uma *string* definida, por exemplo, ou *fill parent* para ocupar todo o espaço

permitido pelo elemento que o encapsula. A Figura 12 mostra as funcionalidades mencionadas.

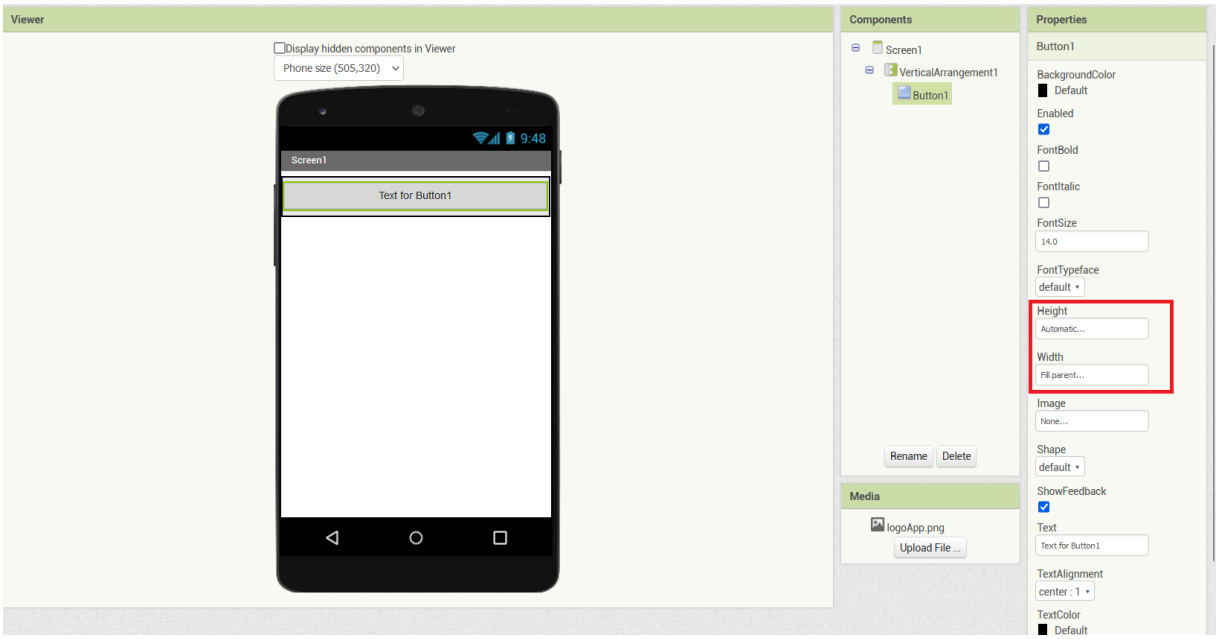


Figura 12. Definição de altura e largura, com largura *fill parent.* (Fonte: próprio autor).

2.3 *Deep learning*

Machine learning (ML) é um ramo da inteligência artificial (IA) e da ciência da computação que se concentra no uso de dados e algoritmos para imitar a maneira como os humanos aprendem, melhorando gradualmente sua precisão (IBM, 2020). Um grande subcampo da área de IA que lida com a habilidade de computadores aprenderem sem ser explicitamente programados (SAMUEL, 1959) conforme apresentado na Figura 13.

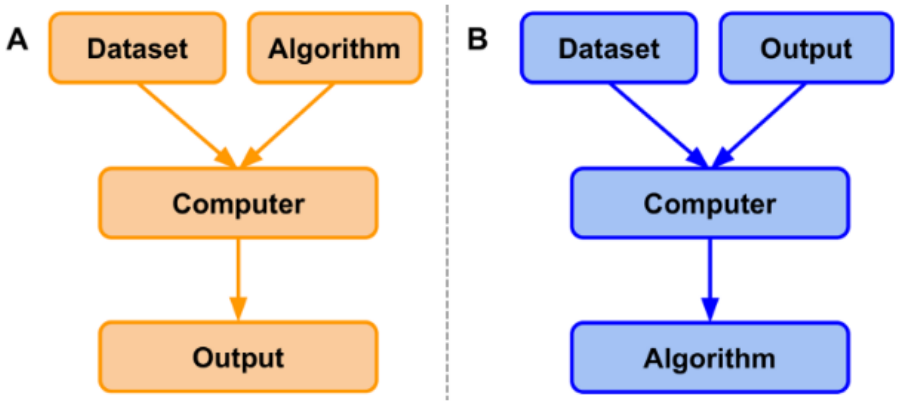


Figura 13 - Modelo clássico de programação vs modelo *machine learning*

(Fonte: CHOI, 2022)

De forma geral, há duas vertentes principais das formas de treinar um modelo ML:

- Aprendizagem supervisionada: dados de treinamento incluem tanto entrada como saída, ou seja, o modelo sabe qual é o resultado esperado e se ajusta conforme erra ou acerta predições. Deve ser capaz de fazer generalizações (resultados corretos para um entrada qualquer sem ter conhecimento a priori do que é esperado) (DONALEK, 2011).
- Aprendizagem não supervisionada: o modelo não possui resultados esperados durante o treinamento. Pode ser usado para classificar dados em conjuntos ou *clusters* de dados relacionados de alguma forma (DONALEK, 2011).

Deep Learning (DL) é um subcampo do ML de aprendizado supervisionado que permite modelos computacionais compostos de múltiplas camadas de processamento aprender representações de dados com múltiplos níveis de abstração (LECUN et al, 2015). Estes métodos melhoraram o estado da arte em reconhecimento de fala, reconhecimento visual de objetos, detecção de objetos e muitos outros domínios como descobrimento de drogas e genética (LECUN et al, 2015).

Uma arquitetura *deep learning* é uma pilha de múltiplas camadas de módulos simples sujeitos a aprendizado, muitos computando mapeamentos não lineares entre

entrada e saída, objetivando melhorar a seletividade (reconhecer aspectos relevantes de uma imagem para classificação de um objeto) e invariância (reconhecer aspectos irrelevantes de uma imagem para classificação de um objeto), por exemplo o fundo da imagem, sua posição na tela, rotação, etc..

A Figura 14 ilustra os relacionamentos entre disciplinas da IA, mostrando também que *deep learning* é um subconjunto de *feature learning* englobado por ML e IA.

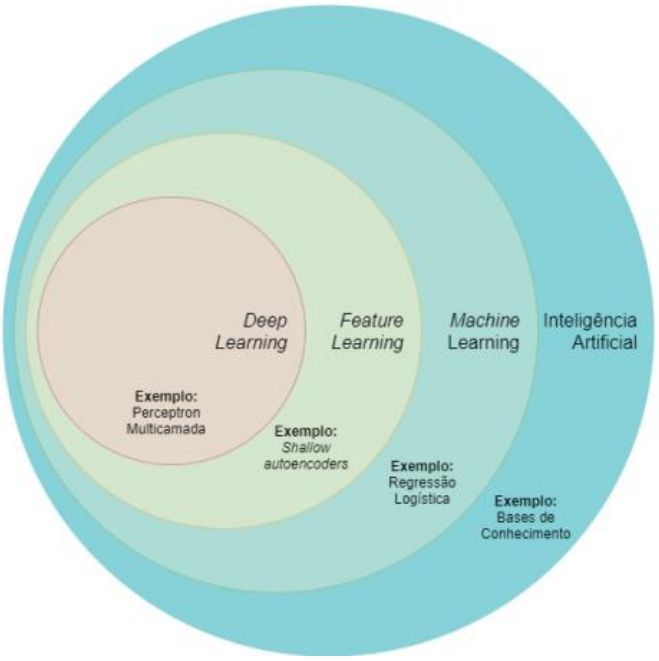


Figura 14. Diagrama dos relacionamentos entre disciplinas de Inteligência artificial
(Fonte: GOODFELLOW, 2016)

2.3.1 Redes Neurais

Uma Rede Neural (*Neural Network*) é um modelo que é projetado para se aproximar da maneira como o cérebro realiza uma tarefa particular ou função de interesse (HAYKIN, 2007). A Figura 15 mostra o esquema de uma rede neural.

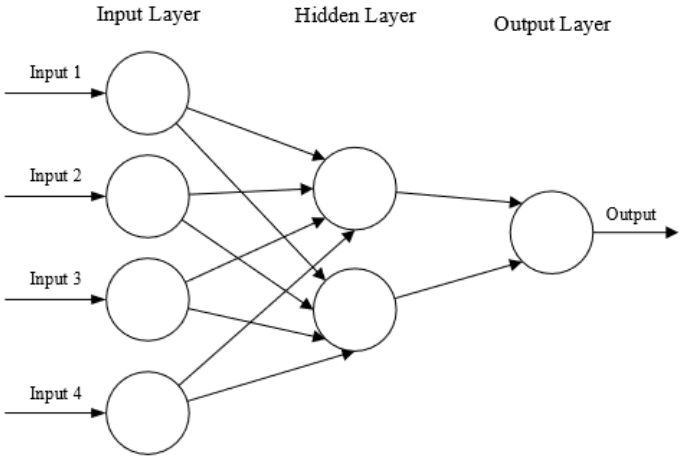


Figura 15. Diagrama de exemplo de uma rede neural (O'SHEA, 2015).

Redes neurais são compostas por nós ou neurônios, conectadas por ligações direcionadas, que servem para propagar a ativação dos nós anteriores (RUSSEL; NORVIG, 2009). Cada ligação tem um peso numérico associado, que determina a força e o sinal de conexão (RUSSEL; NORVIG, 2009). A entrada, geralmente em formato de vetor multidimensional, é carregada na camada de entrada, que a distribui para as *hidden layers*. As *hidden layers* fazem decisões a partir das camadas anteriores e avaliam como uma mudança nos próprios pesos influencia a saída final, quando o modelo utiliza múltiplas *hidden layers* empilhadas, este é comumente chamado de ***deep learning*** (O'SHEA, 2015). As computações são feitas por meio da propagação de sinais nestes nós, e o aprendizado se dá por meio da atualização dos pesos.

2.3.2 *Convolutional Neural Networks (CNN)*

Visão Computacional é o campo da computação que busca técnicas de extrair informações de imagens, vídeos ou qualquer recurso visual (LOGUNOVA, 2022). Nos últimos anos, técnicas de aprendizado profundo vêm fazendo grande progresso no campo de visão computacional para extração de uma estrutura complexa e construção interna de uma representação a partir de entradas sensoriais ricas (YOO, 2015). CNNs utilizam arquiteturas profundas para aprender *features* (informações visuais significativas) diretamente a partir de valores de pixels (LOGUNOVA, 2022). A primeira camada convolucional filtra os *pixels* na imagem para criar um mapa de

features, o mapa é utilizado de entrada para a próxima camada de filtros, produzindo mais um mapa e propagando dessa maneira nas próximas camadas, até a última camada produzir uma classificação ou outra forma de saída (LOGUNOVA, 2022).

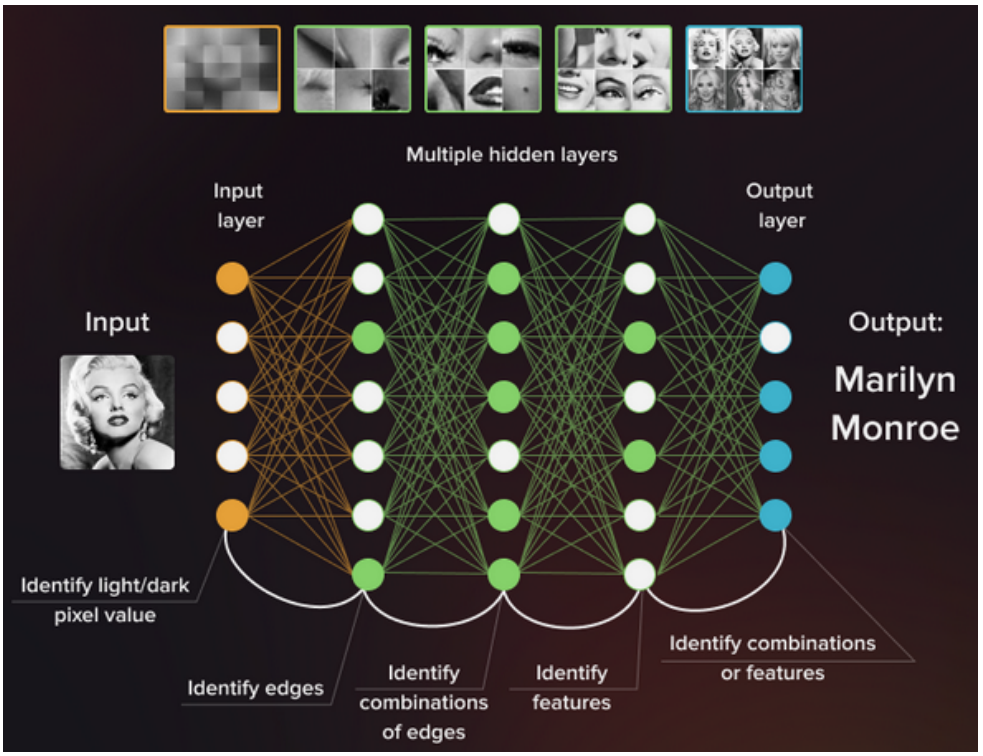


Figura 16. Identificação de *features* numa Rede Neural Profunda.(Fonte: LOGUNOVA, 2022)

Redes neurais convolucionais são análogas a redes neurais no sentido de serem compostas de neurônios que se otimizam por meio de aprendizado. São primariamente utilizados para resolver difíceis problemas relacionados a reconhecimento de padrões em imagens (O'SHEA, 2015).

Uma razão para a utilização de CNNs para resolver problemas relacionados à classificação de imagens é o *overfitting* que ocorre naturalmente utilizando modelos convencionais. *Overfitting* é quando o modelo não consegue generalizar classificações para entradas fora do *conjunto de dados* de treino. Uma arquitetura CNN é composta de três tipos de camadas. São elas camadas convolucionais, camadas de *pooling* e camadas *fully-connected* (O'SHEA, 2015) como mostra a Figura 17.

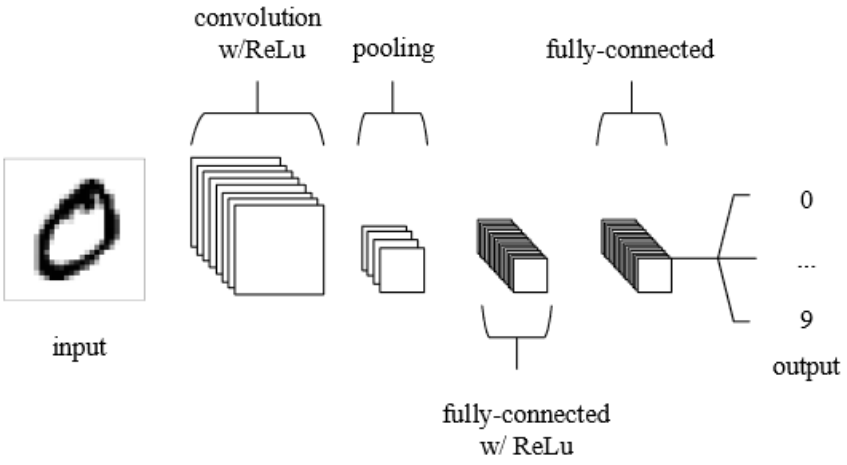


Figura 17. Arquitetura CNN, composta de 5 camadas (Fonte: O'SHEA, 2015)

CNN é classificada como aprendizagem supervisionada. Aplicar aprendizagem supervisionada para um problema de classificação começa na coleta de dados e rotulação, anotando a categoria de cada dado. Em seguida a máquina é treinada a partir destes dados. O treino consiste na máquina atribuir um vetor de pontuações para cada imagem no conjunto de dados de treino. Tal vetor representa o grau de certeza do modelo em classificar determinada entrada em uma certa categoria, então é definida uma função de erro que determina o quão distante o saída está do esperado. A rede neural modifica seus parâmetros internos (pesos) de forma a reduzir os erros. Esta modificação nos pesos em geral é computada por meio de *gradient descent,* calculando a derivada da função de erro no ponto da previsão e computar um novo ponto na direção oposta ao gradiente (minimizando a função) (LECUN et al, 2015)

Muitas aplicações práticas de ML utilizam classificadores lineares com *features*, que são propriedades ou características mensuráveis dos dados (BISHOP, 2006) escolhidas à mão. Entretanto para problemas de classificação mais complexos é interessante um modelo que aprenda boas *features* sozinho e essa é a vantagem do *deep learning* (LECUN et al, 2015).

Camadas convolucionais são responsáveis por aplicar uma operação linear que envolve a multiplicação de um conjunto de pesos com a entrada, a multiplicação é entre a matriz de entrada, geralmente uma imagem, e uma matriz de pesos, chamada de filtro ou *kernel.* O *kernel* tem tamanho menor que os dados de entrada e

as multiplicações ocorrem iterativamente utilizando pedaços da entrada, o resultado é somado, chamado de produto escalar. Esta aplicação do mesmo filtro é proposital pois reduz o número de pesos necessários no treinamento, além de garantir melhor invariância ao modelo, detectando *features* em qualquer lugar da imagem (BROWNLEE, 2020). O processo é exibido na Figura 18.

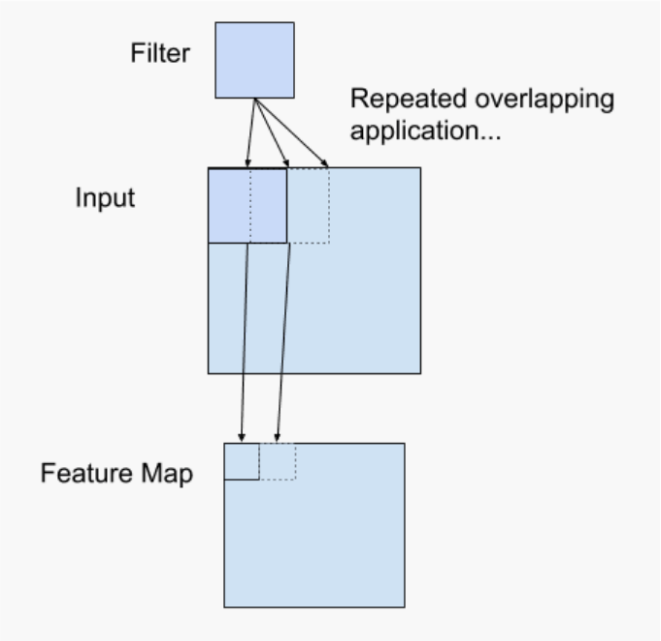


Figura 18. Exemplo de um filtro (*kernel*) aplicado a um entrada bi-dimensional para criar um *feature map.* (Fonte: BROWNLEE, 2020).

O *Feature map* é o resultado das aplicações sobrepostas do filtro sobre o entrada e uma vez que aquele foi calculado, podemos passar cada valor por uma função não linear como ReLU (BROWNLEE, 2020). O ReLU visa aplicar uma função de ativação como uma sigmóide em cada elemento do *feature map* (O'SHEA, 2015).

Já as camadas de *pooling* têm como objetivo reduzir a dimensionalidade da representação, reduzindo o número de parâmetros e a complexidade computacional do modelo. Por fim, camadas *fully-connected* são análogas a neurônios de redes neurais convencionais, conectando diretamente neurônios de camadas adjacentes sem utilização de filtros (O'SHEA, 2015).

2.3.3 *Autoencoders*

Um *autoencoder* é um tipo de rede neural de aprendizado não supervisionado usado para aprender representações eficientes de dados, geralmente para diminuir a dimensionalidade dos dados, ignorando dados insignificantes (KRAMER, 1991), melhorando a invariância. *Autoencoders* são compostos de duas partes, uma para codificação e outra para decodificação. O treinamento desse sistema se dá por meio da minimização entre a distância do dado antes de ser processado e depois de passar pelo processo de *encoding* e *decoding* (Figura 19).

A parte do codificador tem como objetivo traduzir a entrada para um vetor de características latentes, e o decodificador justifica essa representação, pois se o sistema consegue inferir o input a partir da codificação em menor dimensões, de fato as características escolhidas para codificação são as mais importantes ou latentes.

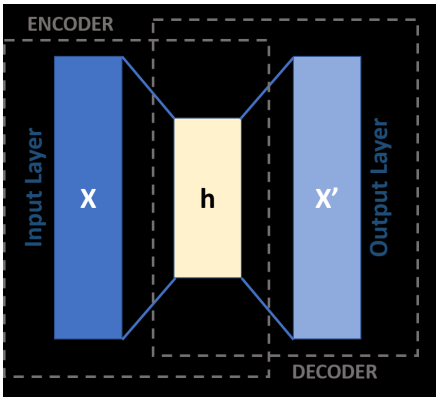


Figura 19. Esquema de um *autoencoder* básico.
(Fonte: MASSI, 2019)

O gargalo (*bottleneck)* é um atributo chave para o design dessa arquitetura. Sem o *bottleneck* de informação, a rede poderia simplesmente aprender a memorizar as variáveis de entrada passando os valores por meio dos nodos (Figura 20).

Existem várias possíveis variações na arquitetura de um *autoencoder.* A forma mais simples é o *autoencoder* incompleto, feito restringindo o número de nodos presentes nas *hidden layers* da rede, limitando a quantidade de informação que pode fluir através desta. *Autoencoder* esparso busca penalizar o uso de nodos das *hidden layers*, isso faz com que o modelo evite memorizar os dados (Figura 20). Essa

abordagem, ao contrário da anterior, permite à rede sensibilizar nodos individuais para detectarem atributos específicos dos dados de entrada, enquanto um *autoencoder* incompleto utiliza a rede inteira para cada observação (JORDAN, 2018)

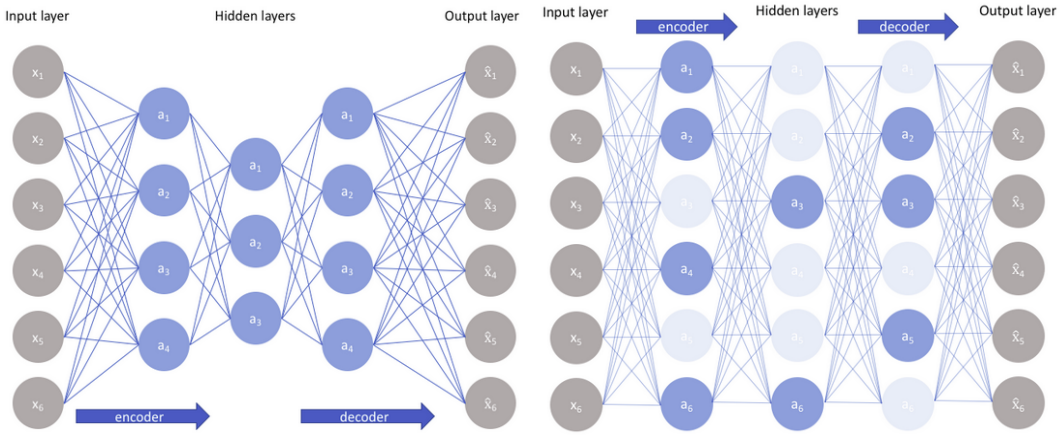


Figura 20. Arquiteturas de *autoencoder* incompleta e esparsa (Fonte: JORDAN, 2018)

Uma vez que o modelo gera uma imagem reconstruída, é feita a comparação dela com a original, computando a diferença e calculando o *loss* que pode ser minimizado (IREKPONOR, 2020). A função de *loss* é descrita matematicamente da seguinte forma:

$$L(\theta, \varphi) = \frac{1}{n} \sum_{i=1}^{n} (x^i - f_\theta(g_\varphi(x^i)))^2$$

(Fonte: IREKPONOR, 2020)

Onde ‘g’ é a função de codificação e ‘f’ de decodificação, a função computa a média das diferenças quadráticas entre a imagem ‘x’ e o produto do modelo.

2.3.4 Técnicas de *clusterização*

Dentro do contexto de resolução de problemas de classificação, muitas vezes são utilizadas técnicas de *clusterização*. *Clusterização* é o agrupamento de conjuntos com características similares. As características são representadas de forma numérica por meio de vetores.

Dentre as técnicas utilizadas neste escopo, pode-se citar *K nearest neighbor* (KNN), com aprendizado supervisionado, para rotular um novo ponto. O algoritmo

analisa os 'k' vizinhos mais próximos ou 'k' pontos de dados mais próximos do novo ponto. Este algoritmo escolhe o rótulo do novo ponto como aquele ao qual pertence a maioria dos 'k' vizinhos mais próximos (SINGH, 2022). Esta técnica somente guarda os pontos e suas classificações, prevendo a classificação de um novo ponto com base na distância euclidiana entre os ‘k’ pontos mais próximos, sem um estágio de treinamento (Figura 21).

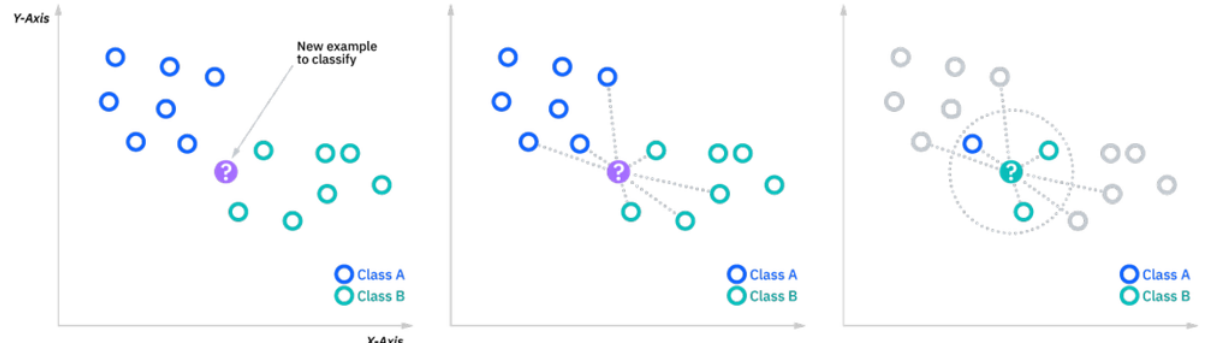


Figura 21. Funcionamento do algoritmo *KNN.* (Fonte: IBM, S.D.)

Já a técnica não supervisionada de *K means* divide os pontos de dados não rotulados em *clusters*/grupos de pontos específicos. Como resultado, cada ponto de dados pertence a apenas um *cluster* com propriedades semelhantes (SINGH, 2022). O número de *clusters* é pré-determinado, e atinge a classificação por meio de uma concepção simples do que é o *cluster* otimizado:

- O centro do *cluster* é a média aritmética de todos os pontos pertencentes ao agrupamento.
- Cada ponto fica mais próximo ao centro do próprio *cluster* do que qualquer outro centro.

Utilizando essas duas suposições existem várias maneiras de chegar na solução. O algoritmo *k-means* de *expectation-maximization* primeiro inicializa pontos centrais de maneira aleatória, em seguida o passo de “expectativa”: associar os pontos ao centro de *cluster* mais próximo (Figura 22). E então o passo de “maximização”: calcular a média de cada *cluster* e mover o centro para lá. Em circunstâncias típicas, cada repetição desses passos deve resultar numa estimativa melhor de agrupamentos (VANDERPLAS, 2016).

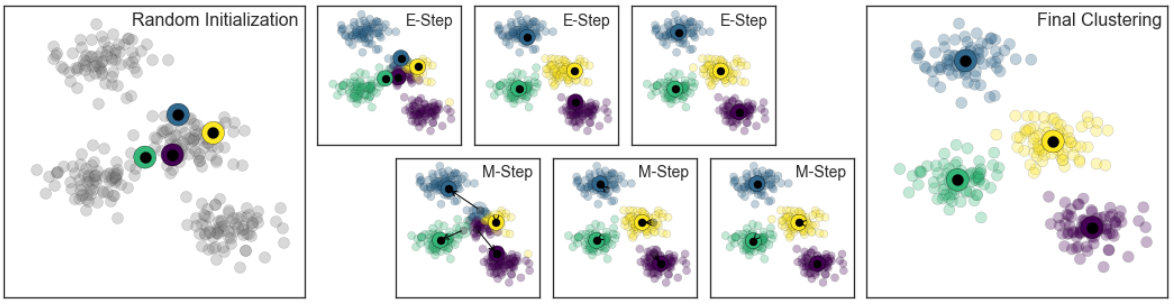


Figura 22. *k-means expectation-maximization* (Fonte: VANDERPLAS, 2016)

Este método apresenta alguns problemas. Entre eles, a posição em que o modelo inicializa os centros de *cluster* podem levar o modelo a convergir em *clusters* não esperados, não atingindo um resultado global otimizado. Outra complicação é o número de *clusters* deve ser escolhido previamente, sendo necessário um algoritmo para determinar esse número ou uma avaliação manual. Os *clusters* formados por meio de *k-means* são limitados linearmente (Figura 23).

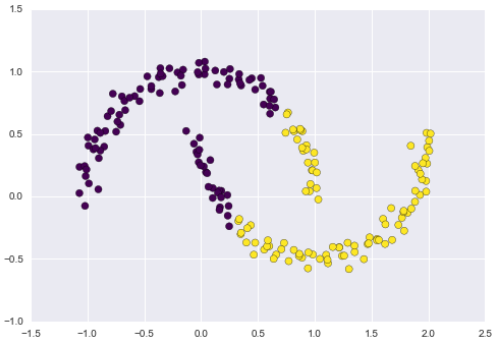


Figura 23. *k-means expectation-maximization*. (Fonte: VANDERPLAS, 2016)

Este último fator negativo do algoritmo pode ser mitigado por meio de uma versão “kernelizada” do algoritmo, computando uma representação dos dados em dimensões maiores, permitindo a separação linear destes (Figura 24).

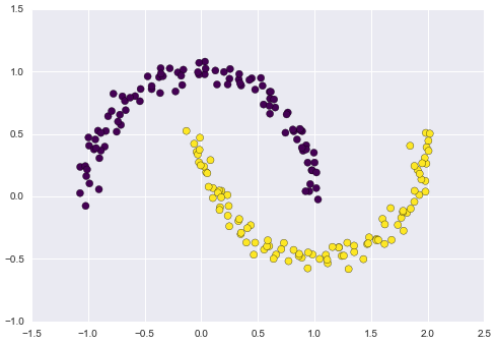


Figura 24. *k-means expectation-maximization* kernelizado. (Fonte: VANDERPLAS, 2016)

2.4 Métricas para a avaliação de desempenho de *machine learning*

O desempenho de um modelo de ML pode ser avaliado de várias maneiras dependendo da tarefa. Algumas das métricas utilizadas nos trabalhos analisados na revisão do estado da arte estão detalhadas na Tabela 3

Tabela 3. Medidas de desempenho em modelos de ML

Tarefa	**Exemplos de métricas de desempenho**
Medida de erro geral	*Mean squared error* (MSE): medida da média dos quadrados dos erros. Geralmente utilizado em processos de treinamento.
Precisão de buscas	*Precision*@K (P@K): para cada busca, são avaliados os K melhores resultados fornecidos por um modelo, é computada a proporção de resultados relevantes e a média para todos as buscas realizadas é a Precision@K (CHEN, 2015).
	Mean Reciprocal Rank (MRR): calculado a partir da recíproca da posição do primeiro elemento relevante retornado por uma busca. Por exemplo, se o segundo elemento retornado é relevante e o primeiro não, x = ½. O MRR é calculado a partir da média de todos os resultados (CRASWELL, 2009)
	Recall: proporção entre elementos relevantes recuperados por um modelo e o total de elementos relevantes.
Correlação entre variáveis	R^2: métrica para medir correlação entre duas variáveis, utiliza como linha de base a média dos dados.
	Pearson correlation coefficient (PCC): mede o grau de correlação entre avaliações de similaridade dados pelo método e pela prova real. Fica no intervalo de -1 até 1. 1 indica que há completa correlação linear entre as variáveis, 0 indica que não há e -1 há uma relação inversa completa.
Identificação de componentes	Intersecção sobre união: ou índice de Jaccard, é um método para quantificar a porcentagem de sobreposição entre a *bounding box* verdadeira e a previsão do modelo
Performance de *clusterização*	Coeficiente de silhueta: baseado na diferença entre ‘A’: distância média entre a amostra e cada elemento e os elementos do *cluster*, e ‘B’: distância média da amostra e de todos os elementos do *cluster* mais próximo. Essa medida busca esclarecer quando um *cluster* está bem definido e propriamente separado de outros.
	Coeficiente de Dunn: calculado com base na divisão entre a menor distância entre *clusters* e o tamanho do maior *cluster*. Se baseia na idéia que uma *clusterização* boa se tem grupos compactos e distantes.

3. Estado da arte

Com o intuito de descobrir técnicas e abordagens já existentes para a detecção de elementos de UI e avaliação da originalidade de um *layout*, utilizando técnicas de *deep learning é realizado um mapeamento sistemático da literatura*. É utilizado o método de mapeamento sistemático definido por Petersen et al. (2015).

Souza et al. (2021) realizou um mapeamento do estado similar, entretanto, a título de comparação, a pesquisa foi feita novamente neste trabalho, com uma *string* de busca diferente e encontrando alguns resultados distintos.

3.1. Definição do Protocolo de Revisão

Pergunta da pesquisa. Quais abordagens existem para a avaliação automática da originalidade de um *layout* de aplicativo Android utilizando técnicas de IA/ML/DL?

A partir dessa pergunta de pesquisa são definidos as seguintes perguntas de análise:

PA1. Que abordagens existem para avaliação de UI de aplicativos Android utilizando *deep learning*?

PA2. Quais são os dados de entrada e saída dessas abordagens?

PA3. Que técnicas de *machine learning* são utilizadas neste contexto?

PA4. Como é feita a preparação dos dados utilizados no treinamento?

PA5. Qual o desempenho das abordagens?

Critérios de inclusão, exclusão e qualidade. Conforme o foco da pesquisa são definidos os seguintes critérios de inclusão e exclusão de artigos:

- São incluídos apenas artigos em inglês;
- São considerados somente artigos que apresentam abordagens para a identificação de padrões em *layouts* no contexto de aplicações com interface de usuário de UI de dispositivos móveis.
- São excluídos artigos que não sejam baseados no aprendizado de máquina envolvendo análise de imagens (ex.: descrição textual).
- São considerados apenas pesquisas publicadas desde 2012 levando em consideração o avanço recente especificamente em relação a aplicativos móveis.

Critérios de qualidade. São considerados apenas artigos que apresentam informações substanciais para se extrair informações referente às perguntas de análise.

Fontes. Foram pesquisados nos principais bancos de dados e bibliotecas digitais no campo da computação, incluindo as Bibliotecas Digitais ACM, a IEEE Xplore e Scopus com acesso por meio do Portal Capes. A pesquisa também foi feita dentro do site Google Scholar para complementar a busca (Haddaway et al., 2015; Piasecki et al. 2017).

Termos de busca. Com base na questão de pesquisa, várias pesquisas informais foram realizadas para calibrar a *string* de busca, identificando termos de pesquisa relevantes e seus sinônimos (Tabela 4). Foram utilizados sinônimos para minimizar o risco de omitir trabalhos relevantes.

Tabela 4. Termos de busca.

Termo chave	Sinônimo	Tradução (inglês)
App	aplicação, Android	application, Android
Layout	interface de usuário, interface gráfica do usuário, estrutura de arame, esboço	user interface, graphical user interface, layout, wireframe, sketch
Originalidade	similar, distinguir, clone	similar, distinguish, clone
Aprendizado de máquina	aprendizado profundo, rede neural, *nearest neighbors, autoencoder*	deep learning, machine learning, neural network, nearest neighbor, autoencoder

String de busca. Após a definição dos termos e seus sinônimos, definiu-se o *string* de busca genérico :

```
(android OR app*) AND (UI OR GUI OR layout OR wireframe OR sketch) AND ("deep learning" OR DL OR "machine learning" OR ML OR "neural network" OR "nearest neighbors"  OR autoencoder) AND (similar* OR distinguish* OR clone)
```

Os *strings* de busca conforme a formatação de cada repositório estão apresentados na Tabela 5. Para o Google Scholar foi definido um *string* em especial para atender as limitações de opções de busca.

Tabela 5. *Strings* de busca.

Repositório	***Search string***
ACM Digital Library	[[Abstract: android] OR [Abstract: app*]] AND [[Abstract: ui] OR [Abstract: gui] OR [Abstract: layout] OR [Abstract: wireframe] OR [Abstract: sketch]] AND [[Abstract: "deep learning"] OR [Abstract: dl] OR [Abstract: "nearest neighbors"] OR [Abstract: "machine learning"] OR [Abstract: ml] OR [Abstract: "neural network"] OR [Abstract: autoencoder]] AND [[Abstract: similar*] OR [Abstract: distinguish*] OR [Abstract: clone]] AND [E-Publication Date: (01/01/2012 TO 10/31/2022)]
IEEE Xplore Digital Library	("Abstract":app* OR "Abstract":android) AND ("Abstract":UI OR "Abstract":GUI OR "Abstract":layout OR "Abstract":wireframe OR "Abstract":sketch) AND ("Abstract":"deep learning" OR "Abstract":DL OR "Abstract":"machine learning" OR "Abstract":ML OR "Abstract":"neural network" OR "Abstract":"nearest neighbors" OR "Abstract":autoencoder) AND ("Abstract":similar* OR "Abstract":distinguish* OR "Abstract":clone) Filters Applied: 2012 - 2022
Scopus	(TITLE (android OR app*) AND TITLE-ABS-KEY (ui OR gui OR layout OR wireframe OR sketch) AND TITLE-ABS-KEY ("deep learning" OR dl OR "machine learning" OR ml OR "neural network" OR "nearest neighbors" OR autoencoder) AND TITLE-ABS-KEY (similar* OR distinguish* OR clone)) AND PUBYEAR > 2011 AND (LIMIT-TO (SUBJAREA , "COMP"))
Google Scholar	"user interface" "layout" “similarity” "machine learning" "neural network" "android" "mobile application"

3.2. Execução da busca

A busca foi realizada em Julho de 2022 pelo autor do trabalho e revisada pela orientadora. A busca inicial resultou em 876 artigos (Tabela 6).

Tabela 6. Número de artigos identificados por repositório e por fase de seleção.

Fonte	**No. de resultados da busca**	**No. de resultados analisados**	**No. de documentos potencialmente relevantes**	**No. de documentos relevantes**
ACM	54	54	13	9
IEEE	99	99	5	2
SCOPUS	83	83	8	4
Google Scholar	640	200	6	4
Total (sem duplicatas)				**10**

A partir do resultado inicial das buscas, foram selecionados artigos potencialmente relevantes de acordo com os critérios de inclusão e exclusão por meio de uma análise do título, resumo e palavra-chave de cada artigo, com a finalidade de confirmar a relevância dos trabalhos de acordo com os critérios de

inclusão e exclusão. Foram analisados ao máximo os primeiros 200 artigos encontrados em cada busca.

Artigos como os de Nguyen (2021) e Baulé (2021) trabalham com vetorização de *layouts* de aplicativos e aprendizado de máquina, entretanto são voltados para a geração de *layouts* e foram excluídos por não lidarem com o agrupamento de *layouts* similares. Artigos como Nguyen et al. (2018) foram excluídos por utilizarem como entrada descrição textual.

Em seguida, foram analisados os artigos potencialmente relevantes pelo texto na íntegra. Como resultado final foram identificados 10 artigos relevantes (Tabela 8).

3.3. Análise dos dados

Para responder às questões de pesquisa, informações relevantes foram extraídas dos 10 artefatos relevantes encontrados. As informações relevantes a serem extraídas estão especificadas na Tabela 7.

Tabela 7. Especificação dos dados extraídos.

Pergunta de análise	**Item**	**Descrição**
PA1. Que abordagens existem para avaliação de similaridade de UI de aplicativos Android utilizando *deep learning*?	Referência	Indicando a referência do artigo
	Nome	Indicando o nome do artigo
	Breve descrição	Descrição breve da proposta do artigo
	Plataforma	Android, IOS, App Inventor
PA2. Quais são os dados de entrada e saída dessas abordagens	Dados de entrada	Tipo de dados utilizados na entrada
	Formato dos dados de entrada	Formato de dados da entrada
	Dados de saída	Tipo de dados que o sistema retorna
	Formato de dados de saída	Formato dos dados de retorno
PA3. Que técnicas de *machine learning* são utilizadas neste contexto?	Técnicas de ML	Que técnica é usada para a identificação de elementos do *layout*
	Técnicas para *clusterização*	Que técnica é usada para agrupamento e comparação de *layouts*
	Tipo de aprendizagem	É utilizado aprendizado de máquina supervisionado ou não supervisionado

PA4. Como é feita a preparação dos dados?	Divisão do conjunto de dados	Proporção utilizada na divisão do conjunto de dados para treino e validação
	Universo de referência	De onde são os apps cuja comparação é feita
	Rotulação	De que forma os dados foram etiquetados
PA5. Qual é o desempenho das abordagens?	Fator(es) de qualidade avaliado(s)	Indicando qual o fator de qualidade usado para analisar a abordagem
	Métricas utilizadas	Que métricas foram utilizadas nos estudos
	Resultados da análise	Indicando quais foram os resultados obtidos pela abordagem

Os artigos selecionados foram lidos de forma completa e os dados foram extraídos pelo autor e revisados pela orientadora. Nos casos em que o artigo não apresenta nenhuma informação a ser extraída sobre um determinado dado, a falta desta informação é indicada como não informada (NI).

PA1. Que abordagens existem para avaliação de UI de aplicativos Android utilizando *deep learning*

Foram encontrados 10 artigos que especificam alguma abordagem que avalie similaridade entre aplicativos Android conforme apresentado na Tabela 8.

Tabela 8. Visão geral das abordagens.

Ref	Nome	Breve descrição	Plataforma
(Bunian et al., 2021)	*Vins: Visual search for mobile user interface design*	Abordagem para buscar *layouts* parecidos com uma imagem base.	Android
(Chen et al., 2020)	*Wireframe-based UI Design Search Through Image Autoencoder*	Treinando um *auto encoder* a abordagem permite codificar um *wireframe* de uma busca do usuário e capturas de tela de UIs de aplicações existentes por meio de seus *wireframes* em um espaço vetorial de designs de UI.	Android
(Deka et al., 2017)	*Rico: A Mobile App conjunto de dados for Building Data-Driven Design Applications.*	Desenvolvimento de um repositório de *layouts* de aplicativo Android e desenvolvimento de um modelo de aprendizado de máquina capaz de fazer buscas por *layouts* similares no conjunto de dados.	Android
(Ge, 2019)	*Android GUI search using hand-drawn sketches*	método baseado em *deep learning* para buscar por aplicativos visualmente similares a um *sketch* de input.	Android
(Huang, 2019)	*Swire: Sketch-based user interface retrieval*	Abordagem para buscar *layouts* parecidos com um *sketch* base.	Android
(Li et al., 2021)	*Screen2Vec: Semantic Embedding of GUI Screens and GUI Components*	Geração de representações de GUIs em vetores por meio de aprendizado de máquina não supervisionado.	Android
(Liu et al., 2018)	*Learning Design Semantics for Mobile Apps*	Abordagem de classificação de itens da UI que permite gerar anotações semânticas. Usando os dados anotados foi treinado uma rede neural para distinguir classes de ícones e realizar classificação de *layouts*.	Android

(Tan et al., 2021)	*A Novel Android Malware Detection Method Based on Visible User Interface*	Utilização de técnicas de *deep learning* detectar aplicativos que potencialmente são *malware* por meio de análise de capturas de tela. Utiliza-se um grande universo de referência como base.	Android
(Wu et al., 2019)	*Understanding and Modeling User-Perceived Brand Personality from Mobile Application UIs*	Criação de um sistema para predizer a percepção do público acerca de um aplicativo baseado na sua identidade visual (*layout*, esquema de cores, etc.).	Android
(Xie, 2019)	*User Interface Code Retrieval: A Novel Visual- Representation- Aware Approach*	Método de avaliação de similaridade entre árvores de representação visual de *layouts*.	Android

Nota-se que o interesse nesta área vem aumentando nos últimos anos, com maior parte tendo sido produzida em 2019 (Figura 25).

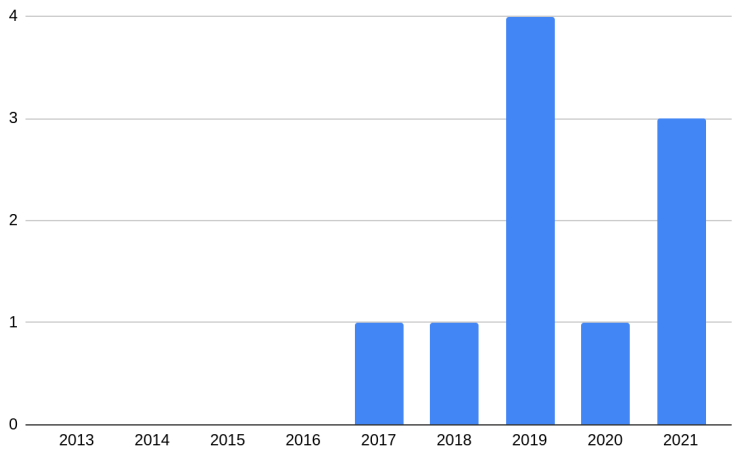


Figura 25. Quantidade de artigos relevantes na busca até a data da busca em 2022. (Fonte: próprio autor)

PA2. Quais são os dados de entrada e saída dessas abordagens?

Grande parte das abordagens utilizam de entradas *screenshots, wireframes* e *sketches.* Muitas buscam criar alguma representação intermediária que indique a hierarquia do layout. Quanto a saída, há uma grande variedade de saídas, entretanto a maior parte dos artigos tem como saída *layouts* parecidos.

Tabela 9. Dados extraídos acerca dos dados de entrada e saída das abordagens

Citação	**Dados**			
	Entrada	**Formato entrada**	**Saída**	**Formato Saída**
(Bunian et al., 2021)	*Screenshot* ou *wireframe*	Imagem	*Screenshots* de *layouts* parecidos	Imagem
(Chen et al., 2020)	*Wireframe*	*Wireframe* desenhada na ferramenta proposta	*Screenshots* de *layouts* parecidos	Imagem
(Deka et al., 2017)	*Screenshots*	Imagem	*Screenshots* de *layouts* parecidos	Imagem
(Ge, 2019)	*Sketch*	Imagem	Código de *apps* com layouts parecidos	NI
(Huang, 2019)	*Sketch*	Imagem	*Screenshots* de *layouts* parecidos	Imagem
(Li et al., 2021)	*Screenshots*	imagem	*Screenshots* de *layouts* parecidos	Imagem
(Liu et al., 2018)	*Screenshots* e hierarquia da UI	Imagem	*Screenshots* de *layouts* parecidos	Imagem
(Tan et al., 2021)	*Screenshots*	Imagem	Classificação binária (é *malware*?) e nível de confiança	Números em Ponto flutuante
(Wu et al., 2019)	*Screenshots*	Imagem	Previsão de pontuação de satisfação com identidade visual de um app	Números em Ponto flutuante
(Xie, 2019)	*Sketch*	Imagem	*Layouts* parecidos	Código

Como esperado por ser critério de exclusão na pesquisa, todos os resultados possuem como entrada algum tipo de imagem, num formato comum de pixels, alguns sendo rgb ou em preto e branco. A entrada mais comum são *screenshots* de *layout*, há também utilização de *wireframes* quando o modelo busca expor melhor a hierarquia de componentes, além de *sketches* em algumas abordagens.

Como saída, diversos modelos, como se tratam de programas de busca por interfaces, dão como saída *screenshots* de *layouts* parecidos. Tan et al. (2021) e Wu et al. (2019) se diferenciam por darem como saída uma decisão, no primeiro caso, identificando um aplicativo como *malware* ou não, e no segundo caso, indicando um grau de satisfação previsto com o *layout* de um aplicativo.

Muitas abordagens como a de Ge (2019), Xie (2019), Liu et al. (2018), Li et al. (2021) entre outras, acabam optando por extrair uma representação intermediária

que melhor expõe o significado dos elementos de interface (Figura 26) ou a hierarquia de um *layout*, para então sobre essa representação realizar o treinamento de um modelo.

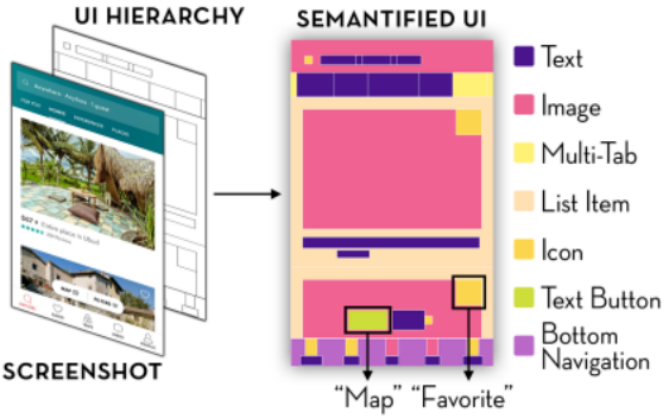


Figura 26. Anotação semântica de UI. (Fonte: Liu et al., 2018)

PA3. Que técnicas de *machine learning* são utilizadas neste contexto?

Metade dos estudos utiliza formas de *clusterização* como *k nearest-neighbours*, uma abordagem simplificada que classifica entradas dentre *rótulos* fornecidas e *k-means* que calculam *clusters* automaticamente.

Tabela 10. Dados extraídos para responder que técnicas de aprendizado de máquina foram utilizados nos modelos

Citação	**Técnicas de ML**	**Técnica de *clusterização***	**Tipo de aprendizado**
(Bunian et al., 2021)	YOLO e *Wireframe autoencoder* (CNN)	Distância euclidiana do vetor de *Features*	supervisionado
(Chen et al., 2020)	*Wireframe autoencoder (*CNN)	KNN	não supervisionado
(Deka et al., 2017)	*Layout autoencoder* (rede neural)	KNN	não supervisionado
(Ge, 2019)	*Deep learning*	*Largest weighted common subtree embeddings (LWCSE)* para *clusterização*	supervisionado
(Huang, 2019)	VGG-A (CNN)	KNN	supervisionado

(Li et al., 2021)	*recurrent neural network model (RNN)*	KNN	supervisionado
(Liu et al., 2018)	*Layout autoencoder* (CNN)	KNN	não supervisionado
(Tan et al., 2021)	VGG19, ResNet50 e MobileNETv2 (CNN)	Problema de decisão binária (não *clusteriza*)	supervisionado
(Wu et al., 2019)	*multiple linear regression (MLR), LASSO linear regression (LLR), multiple-layer perceptron (MLP), decision tree (DT), random forest (RF)*	*K-means*	supervisionado
(Xie, 2019)	CNN	*Layout edit distance*	supervisionado

Chen et al. (2020), por meio de *wireframes*, treina um *autoencoder*. A representação em vetores é utilizada para medir o grau de similaridade de *layouts* de UI. Liu et al. (2018) também utiliza *autoencoder*, que processa uma anotação semântica das UIs para representá-las num espaço vetorial.

O modelo de Li et al. (2021) cria uma representação vetorizada dos domínios do *layout* completo, dos componentes separados e também da descrição do app, juntando todos no final (Figura 27).

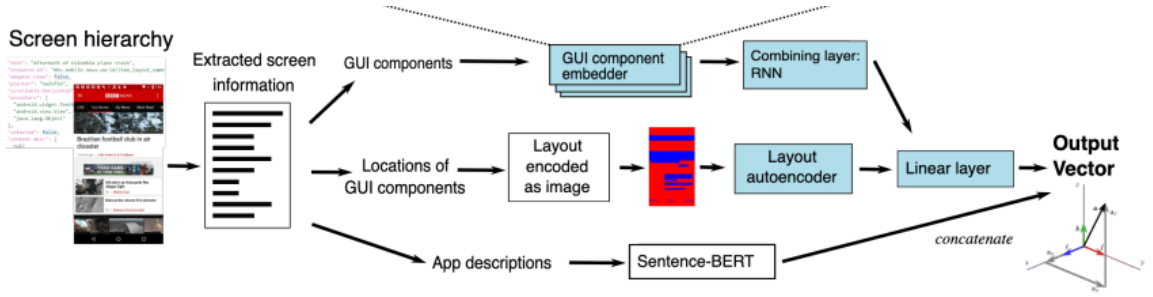


Figura 27. Arquitetura multi-domínio. (Fonte: Li et al., 2021)

Deka et al. et al. (2017) utiliza treinamento de *autoencoder* que permite dar como entrada uma imagem, realizando a codificação do *layout* numa representação de 64 dimensões que pode ser comparada com imagens previamente codificadas do conjunto de dados (Figura 28).

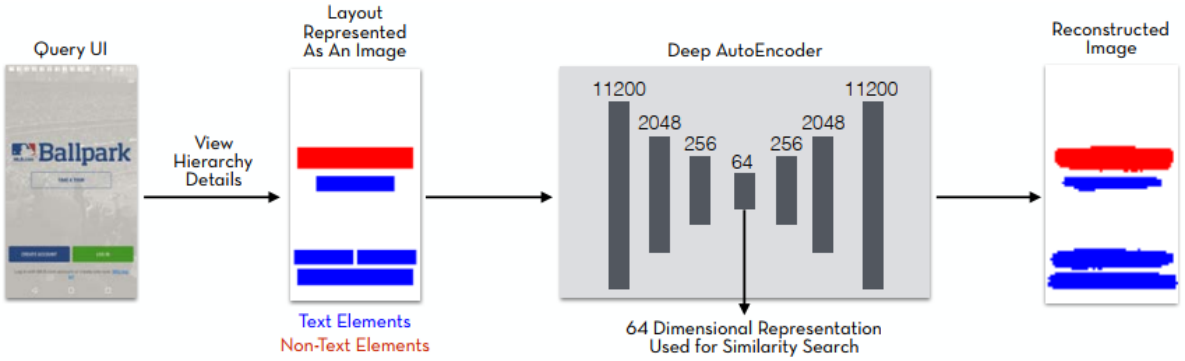


Figura 28. Arquitetura *autoencoder*. (Fonte: Deka et al., 2017)

Ge (2019) utiliza uma entrada em formato de *sketch* e realiza a transformação desta por meio de *machine learning* para uma estrutura intermediária JSON, de forma a criar uma hierarquia entre os elementos de interface, podendo então realizar a busca e comparação entre aplicativos por meio do algoritmo de maior subárvore comum (Figura 29).

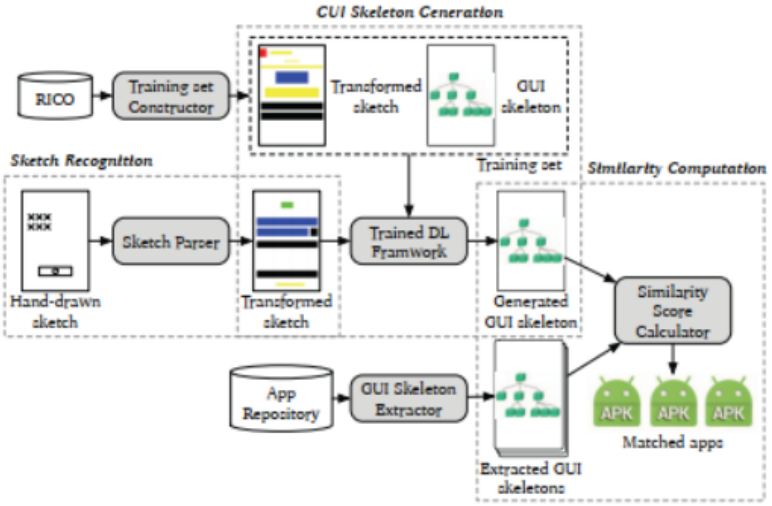


Figura 29. Modelo de busca de *layouts*. (Fonte: Ge, 2019)

Xie (2019) adapta o modelo de pix2code (BELTRAMELLI, 2018) para a geração de um modelo hierárquico intermediário em código. Então o código é convertido em um modelo de árvores que permite sua comparação.

Huang (2019) treina duas redes neurais convolucionais, uma para codificar *screenshots* do conjunto de dados e outra para codificar *sketches*, aplicando uma função de minimização que busca diminuir a distância entre as representações

geradas pelas redes, melhorando o desempenho de reconhecer que *sketch* corresponde a qual *screenshot* (Figura 30).

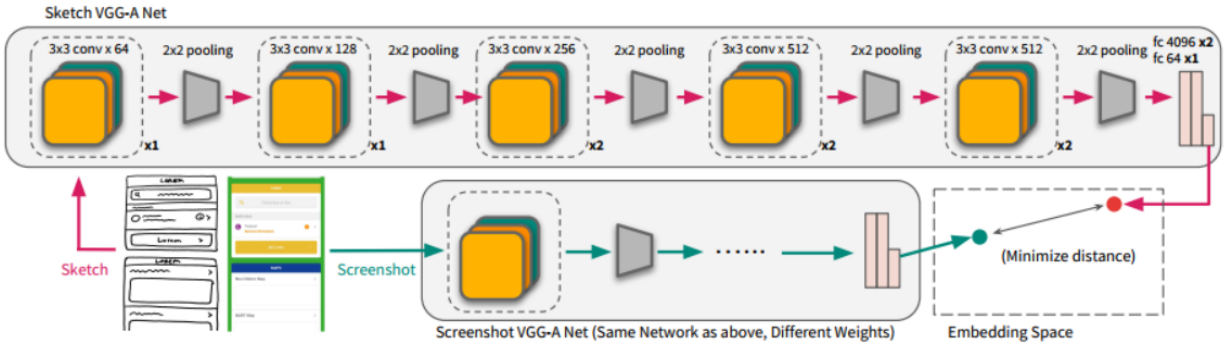


Figura 30. Utilização de duas redes CNN para codificação de *sketch* e *screenshot* separadamente. (Fonte: Huang, 2019)

O método proposto por Bunian et al. (2021) utiliza *Single Shot Multi-box detector*, parecido com YOLO, para reconhecimento de elementos de interface e criação de *bounding boxes*, essa representação intermediária é usada de entrada em um *autoencoder*. No final, a busca é somente uma análise e rankeamento dos *layouts* com menor distância euclidiana do *layout* de *query.*

A análise dos trabalhos relacionados permite afirmar que grande parte dos métodos utiliza uma representação hierárquica intermediária para extrair uma árvore do *layout*, permitindo a sua comparação com o universo de referência. Vários dos trabalhos recorrem a modelos YOLO para detecção de objetos. Outra tendência das abordagens é a criação de uma codificação dos *layouts* de entrada em uma vetor de muitas dimensões, muitas vezes por meio de uma CNN na forma de um *autoencoder* como no caso de Chen et al. (2020), Liu et al. (2018) e Deka et al. (2017), gerando um espaço onde uma técnica de *clusterização*, como KNN, pode ser aplicada, entretanto a maioria das abordagens (sete entre as dez) utilizaram aprendizado supervisionado nesta parte de detecção e representação de objetos.

PA4. Como é feita a preparação dos dados?

A forma que os dados de treinamento foram tratados é apresentada na Tabela 11.

Tabela 11. Tratamento feito nos dados de entrada para os modelos e *conjunto de dados* utilizado.

Citação	**Divisão do conjunto de dados**	**Rotulação**	**Universo de referência**
(Bunian et al., 2021)	divisão em 80:10:10 para treino, validação e teste, respectivamente	Rotulação manual	*conjunto de dados* próprio com 4.800 imagens, incluindo 257 *wireframes* e 4.543 *screenshots*
(Chen et al., 2020)	NI	Não supervisionado	*conjunto de dados* próprio com 54.987 *screenshots* de 7.748 aplicativos
(Deka et al., 2017)	Divisão em 90:10 para treino e validação, respectivamente.	Não supervisionado	Rico *conjunto de dados conjunto de dados*
(Ge, 2019)	NI	NI	Rico *conjunto de dados*
(Huang, 2019)	divisão em aprox. 75:15 para treino e teste, respectivamente	Rotulação manual	subconjunto do Rico conjunto de dados com 2201 UIs de 167 aplicativos
(Li et al, 2021)	Divisão em 90:10 para treino e teste, respectivamente.	Não supervisionado	66,261 *screenshots* não idênticos de 9.384 aplicativos Android da Google Play
(Liu et al., 2018)	Anotação semântica dos elementos e ícones da UI	Utilização de código, componentes são anotados com base em suas classes e as classes das quais são herdeiros	Rico *conjunto de dados*
(Tan et al., 2021)	Divisão em 70:30 para treino e teste, respectivamente	Rotulação baseada no repositório de origem dos apps, aplicativos do AndroZoo como malignos e Google Play como benignos	AnASD *conjunto de dados* com 21.674 *Screenshots* de UI, 11.142 de 4.959 apps benígnos da Google Play e 10.532 de 4.611 apps malignos do AndroZoo
(Wu et al., 2019)	Divisão em 70:30 para treino e teste, respectivamente	Rotulação manual	subconjunto do Rico *conjunto de dados* com 318 aplicativos móveis
(Xie, 2019)	NI	Rotulação manual	conjunto de dados próprio com 6.750 pares de interfaces de usuário

Wu et al. (2019), Xie (2019), Huang (2019) entre outros, realizam o *rotulação* manual, envolvendo em sua maioria um grande número de participantes, mas existem abordagens inteligentes como a de Liu et al. (2018), que utiliza anotação do *conjunto de dados* com base no código, mapeando diretamente informação como nomes das classes dos componentes para seus *rótulos.* Nota-se também que muitas abordagens se tornam mais desafiadoras pela necessidade de rotulação de forma a

produzir um conjunto de dados grande o suficiente para ser estatisticamente relevante.

Deka et al. et al. (2017), Ge (2019), Wu et al. (2019), Liu et al. (2018) e Huang (2019) utilizam o conjunto de dados Rico, que possui em torno de 72.000 *screenshots* únicos de 9.700 *apps* Android. Nota-se que há uma grande utilização deste mesmo conjunto de dados entre os estudos encontrados.

As redes em geral foram treinadas com algo em torno de 70% do conjunto de dados. Muitos dos estudos não separam explicitamente um conjunto de teste e validação, utilizando o mesmo conjunto para ambos os fins.

PA5. Qual é o desempenho das abordagens?

Houve grande variação entre as métricas e técnicas utilizadas para avaliação do desempenho dos modelos (Tabela 12).

Tabela 12. Visão geral sobre as características da avaliação de desempenho

Citação	**Fator(es) de qualidade avaliado(s)**	**Métricas utilizadas**	**Resultados da Análise**
(Bunian et al., 2021)	Precisão	intersecção sobre união, MAP, AUC e P@K	IoU de 74.15% para a média de detecção de todos os componentes, MAP de 76.39%, AUC de 79.02% e P@K com K = 10 de 86.48% e 92.05% para K = 1
(Chen et al., 2020)	Precisão e performance	*runtime*, P@K e MRR	*runtime* de 561.2 s para 500 queries, melhor que muitos outros métodos, melhores resultados utilizando *wireframe* a nível de cores. 70% P@K(k=1) e 0.73 MRR nos treinos comparativos e 44% P@K(k=1) e 0.59 MRR nas avaliações humanas
(Deka et al., 2017)	Precisão	Avaliação empírica dos *layouts* retornados	NI
(Ge, 2019)	Precisão	P@K	Num espaço de 60 aplicativos, foram feitos *sketches* para buscar por 6 aplicativos específicos. O modelo acertou em 3 destes, e nos outros 3 o aplicativo desejado ficou entre os 6 aplicativos que o modelo apontou como mais similares
(Huang, 2019)	Precisão	P@K	Foram realizados 6 *sketches* de aplicativos dentre 60 do universo de referência, o modelo encontrou o resultado exato para 3 destes

			sketches.
(Li et al, 2021)	Precisão	P@K e Avaliação empírica dos *layouts* retornados	P@K de avaliação da predição de componentes, utilizando o modelo com distância hierárquica foi possivel P@K com K = 1 de 0.588. Nas avaliações humanas de 0-5, utilizando a média de notas para 5 *layouts* retornados pelos modelos como mais similares, as precisões foram 3.29, 3.01, 2.41 para as abordagens Screen2Vec, somente texto e somente *layout,* respectivamente
(Liu et al., 2018)	Precisão	Avaliação empírica dos *layouts* retornados	NI
(Tan et al., 2021)	Precisão	*Recall* e Auc	Os melhores resultados foram alcançados por meio do ResNet50, com acurácia 0.77, precisão 0.79, *recall* 0.75, Auc 0.81 no conjunto de validação
(Wu et al., 2019)	Precisão	MSE e coeficiente de determinação (R^2)	os melhores resultados foram alcançados pela abordagem utilizando random forest (RF) com MSE=0.035 e R^2=0.78 (correlação entre previsão e prova real)
(Xie, 2019)	Precisão	Avaliação de *MSE* e *PCC*	O modelo produz um *MSE* de 0.027, 51.8% menor que o melhor competidor e um *PCC* 27.6% maior que o melhor competidor.

Todos os modelos foram avaliados utilizando alguma métrica de precisão, apesar de alguns trabalhos como de Liu et al. (2018) e Deka et al. (2017) não apresentarem métricas formais para a avaliação do modelo de comparação de *layouts*.

Muitos dos trabalhos utilizam métricas como erro quadrático médio e área abaixo da curva. O MSE abaixo de 0.035 atingido nas pesquisas de Xie (2019) e Wu et al. (2019) é considerado uma alta precisão (WU, 2019). Também houveram algumas pesquisas que usaram P@K, buscando medir o quanto o modelo é preciso no que retorna para um *layout* de entrada. Esse tipo de métrica é utilizada por Ge (2019), Bunian et al. (2021) e Chen et al. (2020), os modelos acabaram funcionando relativamente bem segundo essa métrica. Entretanto houve com grande variação, ficando em torno de 40% a 90% para diferentes K. Em geral, os que alcançaram resultados piores foram com K>5. Isso indica que os modelos divergem mais da percepção humana quanto maior o número de resultados avaliados. Outros modelos avaliam a correlação, entre a percepção humana e do modelo, como coeficiente de

relação de Pearson. No caso de Xie (2019) alcança um valor de 0.88, considerado uma correlação forte, levando em conta a comparação com outros métodos analisados no mesmo estudo. Vale ressaltar que outras pesquisas encontradas não utilizam uma métrica de correlação de variáveis.

3.4. Discussão

A partir da revisão sistemática foi possível constatar que existem poucos modelos de avaliação da similaridade de *layouts* de aplicativos. Muitos dos modelos propostos como os de Liu et al. (2018), Chen et al. (2020), Li et al. (2021), Deka et al. (2017), Ge (2019), Huang (2019) e Bunian et al. (2021) propõem um método de busca por *layouts* similares dentro de um conjuntos de dados, entretanto, nenhum artefato encontrado aborda especificamente detecção de *layouts* similares no contexto do App Inventor.

Observa-se uma tendência a utilizar alguma forma de CNN e/ou KNN. Em particular, soluções que juntam *autoencoder* com *clusterização* possuem certo respaldo na literatura para resolver problemas de análise de similaridade de layouts. Há também vários casos de utilização de representações intermediárias como *wireframes* no caso de Liu et al. (2021) e Ge (2019) ou estruturas hierárquicas em forma de árvore como por Xie (2019) e Ge (2019).

A maioria dos métodos, por utilizarem rotulação manual, encareceram o processo de construção do modelo, além de aumentar o tempo de desenvolvimento dos projetos. Por isso, abordagens não supervisionadas podem ser mais viáveis para o trabalho no presente escopo, de forma a não limitar demais o tamanho do conjunto de dados devido à dificuldade de rotulação dos dados.

Observa-se que várias pesquisas utilizaram o conjunto de dados Rico, este possui algumas vantagens significativas como uma amostragem grande de dezenas de milhares de telas de aplicativo e sua licença permite acesso livre aos pesquisadores.

As métricas de precisão que mais foram utilizadas são MSE e P@K. Poderá ser utilizado MSE na parte de treinamento do modelo, e P@K como uma métrica para avaliação da qualidade do modelo na parte de agrupar *layouts* similares. O MSE abaixo de 0.035 relatado por Xie (2019) e Wu et al. (2019) é considerado uma

alta precisão. Entretanto, no final a métrica de qualidade do modelo desenvolvido deve ser a qualidade das avaliações do modelo a partir da percepção humana.

Como apontado na introdução deste trabalho, no contexto da iniciativa computação na escola/INCoD/INE/UFSC já se iniciou pesquisa nessa área, resultando em modelos para detecção de *layout* de telas de apps criados com App Inventor e cálculo de uma nota de originalidade. Souza (2022) utiliza diferentes abordagens para medir similaridade. Uma alternativa é utilizado código como entrada do seu modelo, ela realiza *parsing* da *string* do código, buscando a frequência dos elementos de interface. Então como medida de similaridade, é calculada uma *combined similarity* que leva em conta quão comum é a ocorrência dos componentes de UI do aplicativo de consulta no universo de referência, e o número de combinações de componentes de UI iguais ao app de consulta. Outra abordagem utilizada nesta pesquisa é baseada em redes neurais treinadas de forma supervisionada, de forma a reconhecer elementos de UI e suas posições. Foi feita uma comparação entre as arquiteturas Detectron2 e YoLO V5, também utilizada por Kreuch (2022). Ambos utilizam a abordagem de Mao et al. (2018) para calcular a similaridade dos aplicativos a partir dessas informações extraídas pelo modelo acerca dos elementos de interface.

Por meio do modelo baseado em YOLO, Kreuch (2022) alcançou um mAP de 91.4%. E para comparar os resultados automaticamente gerados com um julgamento humano obteve-se uma Correlação de Spearman, de 0.709 de correlação entre os resultados do modelo e as avaliações humanas, indicando haver uma forte correlação.

Souza (2022), utilizando Detectron2 apresenta como resultado mAP de 29%, apontando inadequação da arquitetura ao compará-la com YOLO neste escopo de pesquisa.

Tanto Kreuch (2022), como Souza (2022) utilizam para o treinamento dos seus respectivos modelos em torno de 300 *layouts* de aplicativo App Inventor rotulados manualmente.

Na parte de classificação por comparação da similaridade de layouts de aplicativos, muitos estudos analisados utilizam uma abordagem menos ligada aos componentes individuais. Abordagens como a de Kreuch (2022) utilizam YOLO para detectar elementos e a partir do método de Mao et al. (2018) calcular similaridade por meio de comparação entre tipos de componentes, coordenadas e largura.

Observa-se que uma série de estudos utilizam *non linear principal component analysis* (*autoencoder*) nos seus modelos por ser uma simplificação do layout como um todo, criando uma representação menos dependente de elementos individuais. Os estudos analisados no estado da arte não utilizam métricas que calculam coeficientes de correlação entre a percepção humana e a percepção do modelo.

Ameaças à Validade da Revisão da Literatura: Como em qualquer mapeamento sistemático, existem possíveis ameaças à validade dos resultados. Algumas ameaças potenciais foram identificadas, sendo aplicadas estratégias de mitigação para minimizar seus impactos:

- Viés de publicação: Resultados positivos têm maior probabilidade de publicação do que resultados negativos, o que representa sempre um possível viés para mapeamentos sistemáticos. Devido à pouca influência que a tendência destes artigos tem sobre este mapeamento sistemático, uma vez que se refere a uma avaliação do estado da arte, este viés não representa uma ameaça grave.
- Identificação de estudos: A omissão de estudos relevantes é outro risco, mitigado por meio da construção cuidadosa de uma string de busca abrangente, por meio da utilização de diversos sinônimos de conceitos relevantes. A utilização de diversas bases de dados também colabora para a mitigação deste risco.
- Seleção e extração de dados de estudos: Ameaças para a seleção e extração de dados foram mitigadas por meio do fornecimento de uma definição detalhada dos critérios de inclusão/exclusão e de qualidade. Foi definido e documentado um protocolo rígido para a seleção do estudo, que foi realizada e revisada cuidadosamente.

4. Modelo de avaliação automática de originalidade da tela de aplicativos desenvolvidos com App Inventor

Neste capítulo é apresentada a proposta de modelo para avaliação da originalidade de layouts de aplicativos Android desenvolvidos com App Inventor.

4.1. Análise de requisitos

O objetivo deste trabalho é o desenvolvimento de um modelo capaz de avaliar a originalidade de layouts de interfaces de usuário de apps produzidos no App Inventor.

O modelo recebe como entrada o *screenshot* da tela de um aplicativo Android, no contexto educacional. Esta captura de tela é tirada de um aplicativo novo criado por um estudante como resultado de seu aprendizado de criação de aplicativos no App Inventor. O modelo avalia a originalidade da tela comparando com um universo de referência. Este universo de referência é um conjunto de imagens utilizadas no treinamento do modelo.

Desta maneira, adotando a notação de desempenho de Mitchell (1997), o objetivo é desenvolver um programa de computador que com uma experiência E aprenda a realizar uma tarefa T, medindo seu desempenho P, de forma que P melhore em T com experiência E. Aqui, a Tarefa (T) é avaliar a originalidade de uma interface de aplicativo Android, a Experiência (E) é adquirida pelo modelo por meio do treinamento, feito sobre um universo de referência de 8.010 imagens de capturas de tela de aplicativos App Inventor, o desempenho P é medido por meio do MSE entre as capturas de tela de entrada e as imagens reconstruídas pelo modelo. A nota é dada a um *layout* de entrada numa escala numérica [0 ... 1].

Para se diferenciar dos trabalhos já existentes decidiu-se por utilizar treinamento de forma não supervisionada, minimizando o erro quadrático entre a entrada de um auto-encoder e a saída do decoder. A nota deverá ser dada com base na distância entre os elementos do *corpus* e o aplicativo de entrada. O desempenho mínimo desejado é de 0.035 (Wu, 2019).

O processo foi subdividido nas seguintes etapas (WANGENHEIM, 2021):

1. Seleção de um conjunto de capturas de tela de um universo de referência para ser utilizados como dados de treinamento e validação.
2. Treinamento de um modelo utilizando auto-encoder, capaz de “resumir” os dados de entrada, descrevendo-os de acordo com suas features mais importantes, realizando uma avaliação de desempenho e comparação com diversos parâmetros de treinamento.
3. Predição: Comparação de uma nova entrada codificada (com o mesmo modelo treinado no passo anterior) com o universo de referência, de forma a calcular o grau de similaridade. Assim, prever uma pontuação contínua da classificação da originalidade num intervalo de 0 a 1 sendo 0=nada original e 1=muito original.
4. Avaliação: Comparação dos resultados do modelo com avaliação humana.

A Figura 31 exemplifica o fluxo idealizado do modelo.

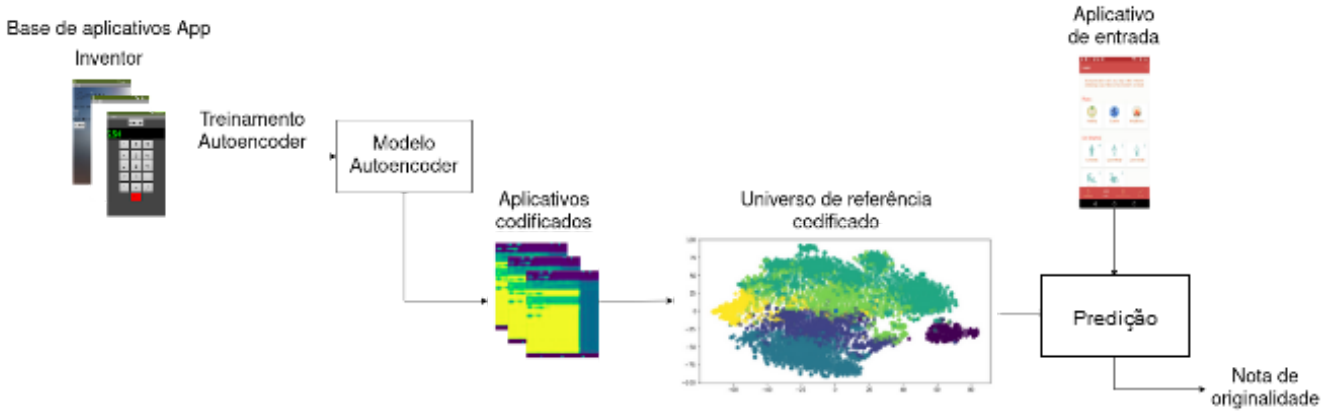


Figura 31. Modelo idealizado (Fonte: próprio autor).

4.2. Preparação do conjunto de dados

A criação do modelo para realização da primeira etapa se iniciou com a busca e preparo do conjunto de dados utilizado.

4.2.1. Conjunto de dados

Levando em conta o foco em aplicativos desenvolvidos por meio de App Inventor, são utilizados os *screenshots* coletados por pesquisadores do

GQS/INCoD/INE/UFSC em Janeiro de 2021. Esses 8.010 *screenshots* foram coletados de 1.551 apps selecionados aleatoriamente de um conjunto de mais de 88 mil apps da galeria do App inventor fornecido pelo MIT. O conjunto de dados coletado possui uma resolução de imagens de 1.080x1.920 pixels. A Figura 32 apresenta exemplos de *screenshots* do conjunto de dados.

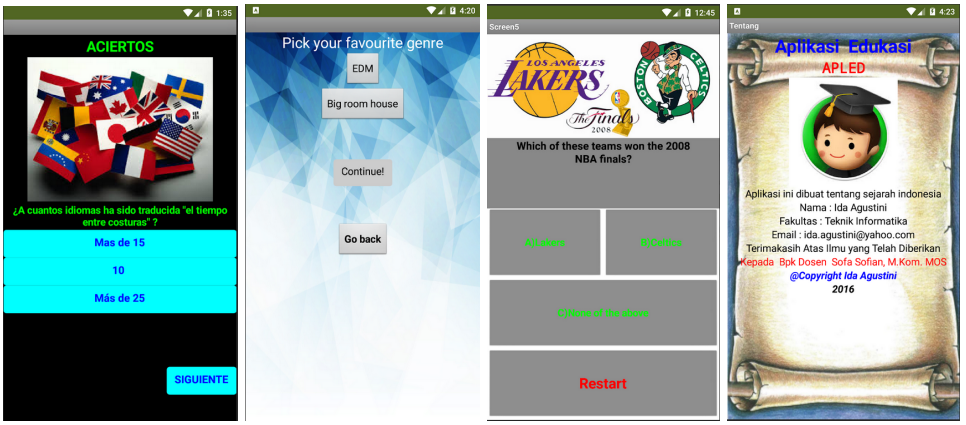


Figura 32. Exemplos de *screenshots* do *dataset* (Fonte: INCoD).

Dentre esses *screenshots,* 85% foram utilizados como universo de referência para treino e 15% como validação.

4.3. Treinamento do modelo de ML

Seleção do Modelo. Foi utilizado o modelo de *autoencoder* para resolver a questão do reconhecimento e sumarização de *features*. Foi encontrado amplo respaldo em pesquisas utilizando *autoencoder* para resolver problemas de classificação de similaridade de aplicativos. Os *autoencoders* são compostos de diversas camadas de convolução e *max pooling* de forma a criar uma representação simplificada de uma entrada, e em seguida reconstrução por meio de camadas de convolução e *up sampling*, medindo o erro entre a entrada e saída para atualização dos pesos.

Foi utilizada a biblioteca Keras do python, que por sua vez emprega Tensor Flow, escrito em python e C++ para alto desempenho, em específico multiplicação de matrizes.

Uma descrição mais detalhada do modelo está presente na Figura 33.

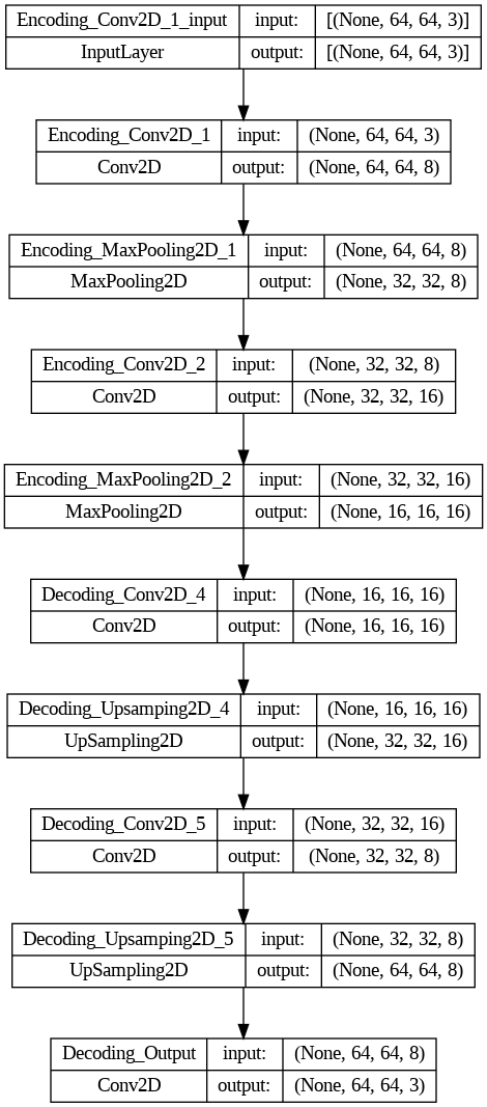


Figura 33. Detalhamento do modelo (Fonte: próprio autor).

Treinamento do modelo e avaliação de desempenho. O modelo utiliza o algoritmo de Adam (KINGMA, 2015), que é uma extensão do algoritmo de gradiente estocástico descendente. A maior diferença entre estes está no algoritmo clássico utilizar um taxa de aprendizagem fixo para todos os pesos durante todo o treinamento, enquanto Adam utiliza taxa de aprendizagem com valores diferentes para cada peso e que seguem uma média móvel exponencial, dando ênfase nas alterações mais recentes no gradiente. Para este algoritmo, uma taxa de aprendizado comumente utilizada é 0.001 (KINGMA, 2015), nos testes do modelo foi sempre utilizado este número como taxa inicial. Foram experimentadas diversas combinações de parâmetros para o modelo, conforme apresentado na Tabela 13.

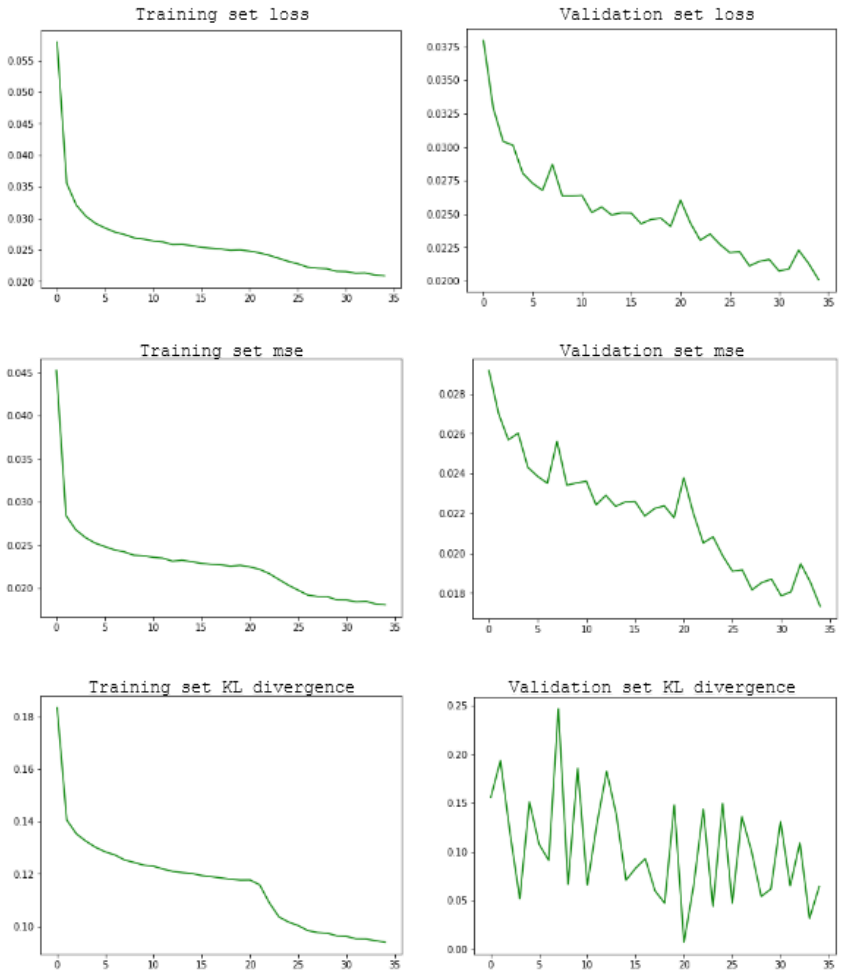
Em comparação com o estado da arte os resultados são bons, com o treinamento atingindo MSE menor que Wu, 2019 e Xie, 2019 em seus respectivos estudos (0.035 e 0.027 respectivamente). Observando que como os melhores resultados foram atingidos no experimento 5, foram utilizados estes parâmetros para o treinamento do modelo conforme especificado na Tabela 14.

Tabela 13. Parâmetros testados e respectivos resultados

Experimento	**Épocas**	**Tamanho do *batch***	**Conjunto de treino**		**Conjunto de validação**	
			MSE	**KL-divergence**	**MSE**	**KL-divergence**
1	25	8	0.0199	0.0965	0.0185	0.0715
2	25	16	0.0214	0.1008	0.0210	0.1447
3	50	16	0.0180	0.0910	0.0176	0.0729
4	16	32	0.0240	0.1185	0.0235	0.1497
4	**35**	**8**	**0.0168**	**0.0823**	**0.0162**	**0.0644**

Tabela 14. Parâmetros finais utilizados no treinamento do *autoencoder*

Quantidade de épocas	35
Tamanho do batch	8
Taxa de aprendizagem	0.001
Quantidade total de imagens usadas para treinamento e validação	8010
Divisão do conjunto de dados	O conjunto de dados foi dividido aleatoriamente em um conjunto de treinamento com 6808 imagens (85%) e um conjunto de validação de 1202 imagens (15%)

Resultados do treinamento e validação	

O treinamento foi realizado utilizando Jupyter Notebook no Google Colab.

Pode notar-se um aumento da performance no conjunto de dados de treinamento e uma tendência de melhora no conjunto de validação. O erro médio quadrático é avaliado no sistema como a divergência entre a entrada e a saída após ser codificada e decodificada, e está interligada ao *loss* utilizado no treinamento do sistema. O treinamento resultou em MSE = 0.0183.

Como análise qualitativa, foram analisados alguns exemplos de imagens produzidas pelo modelo utilizando o conjunto de dados de validação:

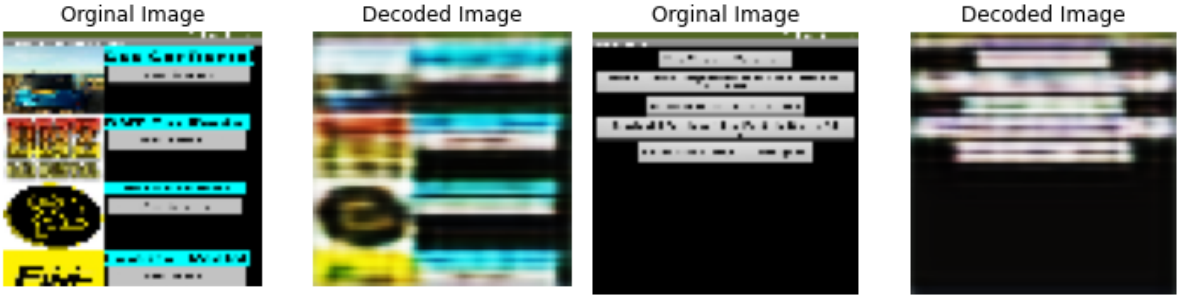


Figura 34. Imagem original e completamente processada pelo modelo (Fonte: próprio autor).

As restaurações parecem boas, talvez um treinamento menos extenso melhore a capacidade de abstrair um pouco mais os detalhes da interface.

Também são demonstradas algumas representações intermediárias das interfaces retiradas da última camada de *downsampling* do modelo na Figura 35.

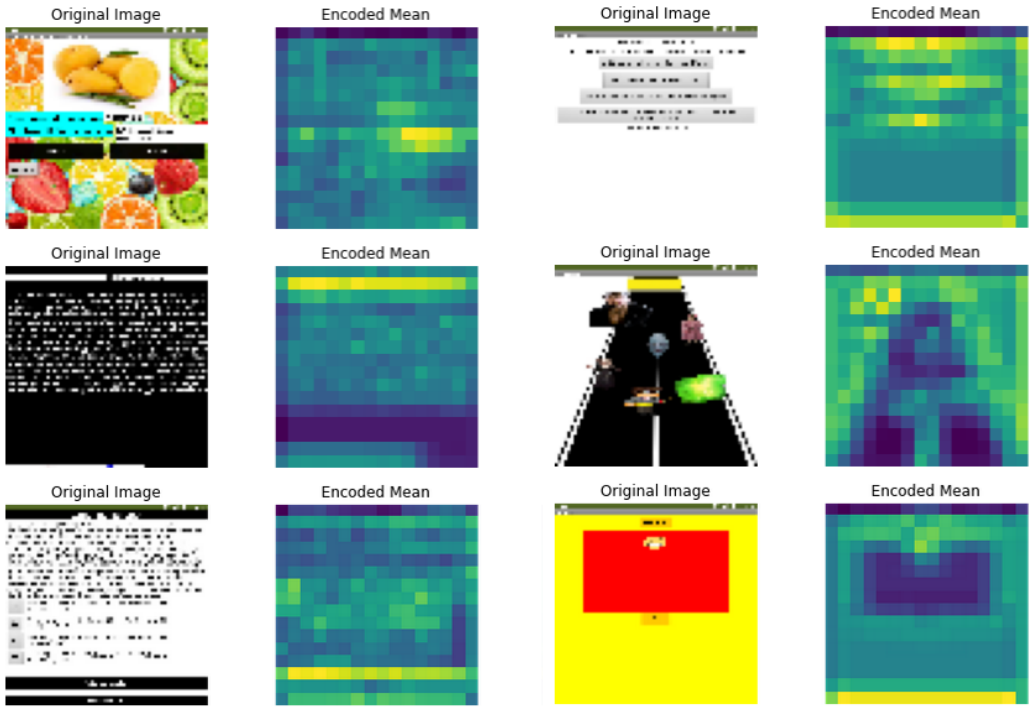


Figura 35. Imagem original e *downsampled* pelo *autoencoder* (Fonte: próprio autor).

Pixels amarelos indicam uma alta ativação dos neurônios, e são os principais pontos que ajudam a diferenciação entre imagens. Nos exemplos percebe-se a presença de alta ativação em elementos de texto, botões e labels.

4.4. Análise de similaridade

Para avaliar o grau de originalidade, realizou-se a clusterização dos dados codificados na etapa anterior, com o propósito de utilizar as correlações entre aplicativos e clusters, a fim de obter uma nota de originalidade.

4.4.1 Cálculo da pontuação por similaridade

Para o cálculo de similaridade foi realizado uma clusterização por meio de k-means utilizando-se como parâmetros para otimização do número de clusters as métricas de cluster discutidas no trabalho de Marcos (2021). A figura 36 contém uma representação que passou por uma redução de dimensionalidade TSNE, para visualização.

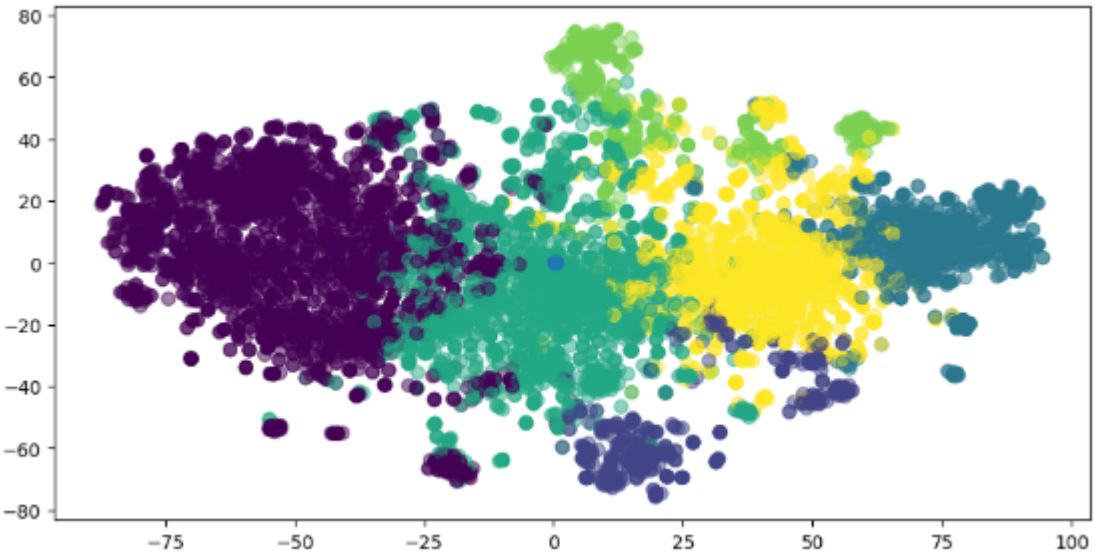


Figura 36. visualização 2d dos clusters (Fonte: próprio autor).

O único método de clusterização testado foi o K-means por não utilizar supervisionamento (classificação manual), entretanto a pesquisa se beneficiaria de mais estudo e experimentação nessa área.

O único parâmetro necessário para treinar o modelo é o número de *clusters*. O método para escolha de número de agrupamentos foi utilizando como critério as métricas de calinski harabasz, *silhouette coefficient*, e davies bouldin, e treinando o modelo de classificação com múltiplas opções de número de clusters num intervalo de 2-18. Ao final foi escolhido o número 7 como número de *clusters* do modelo, pois

analisando os gráficos, nota-se que para algumas métricas a pontuação acaba decaindo rapidamente após 7 *clusters* isso se deve ao fato de existirem muitos parâmetros nos vetores que representam as capturas de tela, tornando os agrupamentos menos bem definidos.

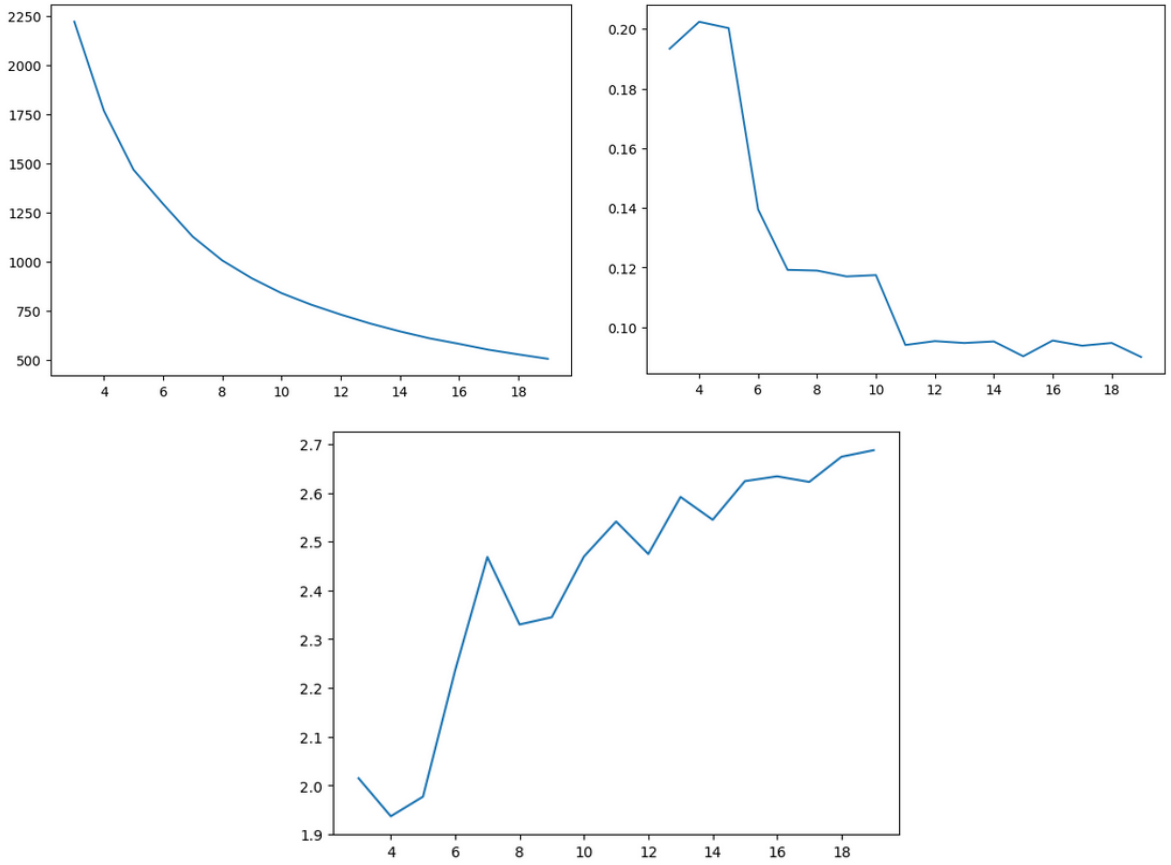


Figura 37. Pontuação do modelo avaliado utilizando métricas de calinski harabasz, *silhouette coefficient*, e davies bouldin, respectivamente, para K clusters num intervalo 2-18 clusters (Fonte: próprio autor).

Figura 38. alguns exemplos retirados de clusterização com k = 7 (Fonte: próprio autor).

A fórmula para o cálculo da nota de originalidade foi baseada na distância entre a representação codificada de uma captura de tela de aplicativo e o seu respectivo cluster, assim como a média da distância desta com os outros clusters, levando em conta todos as outras capturas de tela de aplicativo.

A seguinte fórmula descreve o cálculo:

$$y = d(x, cluster(x)) + 1/n \sum_{n}^{i=1} d(x, c[i]))$$

A função ‘d’ calcula a distância euclidiana entre seus dois parâmetros, ‘x’ é a representação codificada do layout, ‘cluster’ é uma função que calcula o centro do cluster com base no cluster que o ponto ‘x’ pertence, ‘n’ é o número de clusters e ‘c’ é o vetor de clusters.

A distribuição das notas de similaridade de aplicativo (Figura 39) dadas pelo modelo. As notas foram normalizadas para um intervalo entre 0 e 1. Analisando a distribuição no universo de referência composto de 8.010 screenshots, como esperado, existe uma curva que indica a presença de muitos aplicativos que apresentam características semelhantes, ou seja, menos originais, e poucos aplicativos originais.

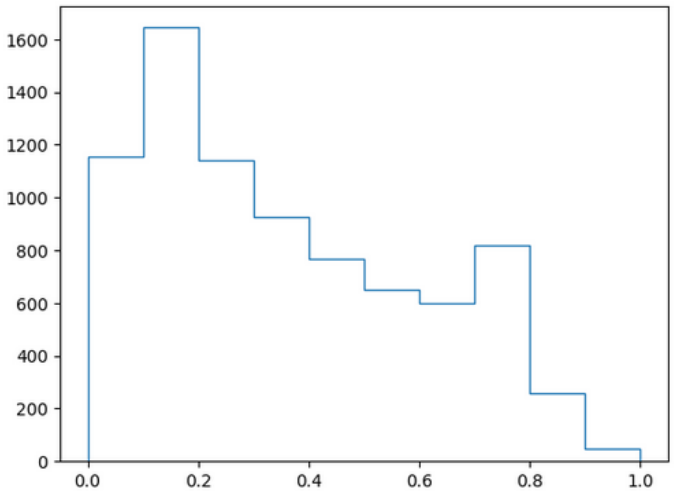


Figura 39. Distribuição de notas de originalidade no universo de referência (Fonte: próprio autor).

Figura 40. Exemplos de capturas de telas consideradas “não originais” pelo modelo e suas respectivas notas (Fonte: próprio autor).

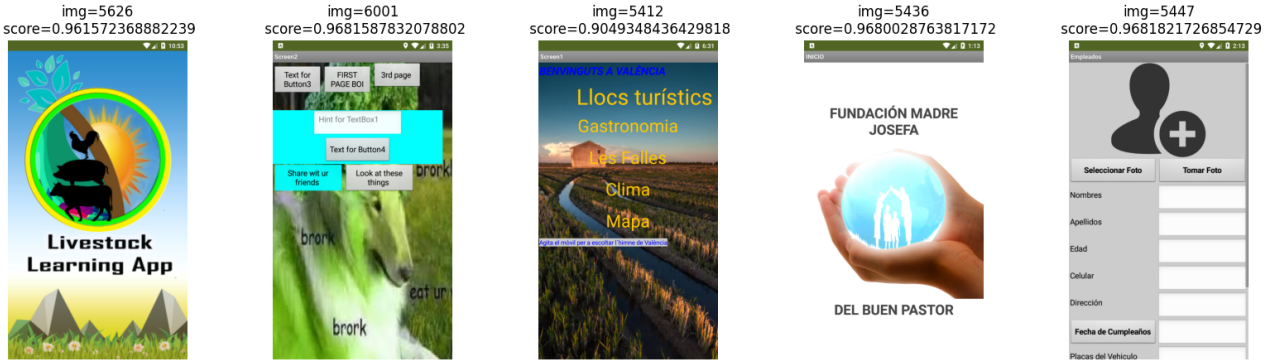


Figura 41. Exemplos de capturas de telas consideradas “originais” pelo modelo e suas respectivas notas (Fonte: próprio autor).

4.5 Avaliação do desempenho

A avaliação de desempenho teve como objetivo validar o modelo, por meio da comparação entre a avaliação automatizada do modelo com a avaliação realizada por humanos. Para tal, foram selecionadas 10 capturas de tela de 10 apps diferentes. Estas 10 telas foram avaliadas por especialistas em relação a sua novidade/originalidade ranking eles de mais original (1) até menos original (10). Em paralelo é calculado automaticamente uma nota de originalidade para cada captura de tela usando o modelo desenvolvido neste trabalho.

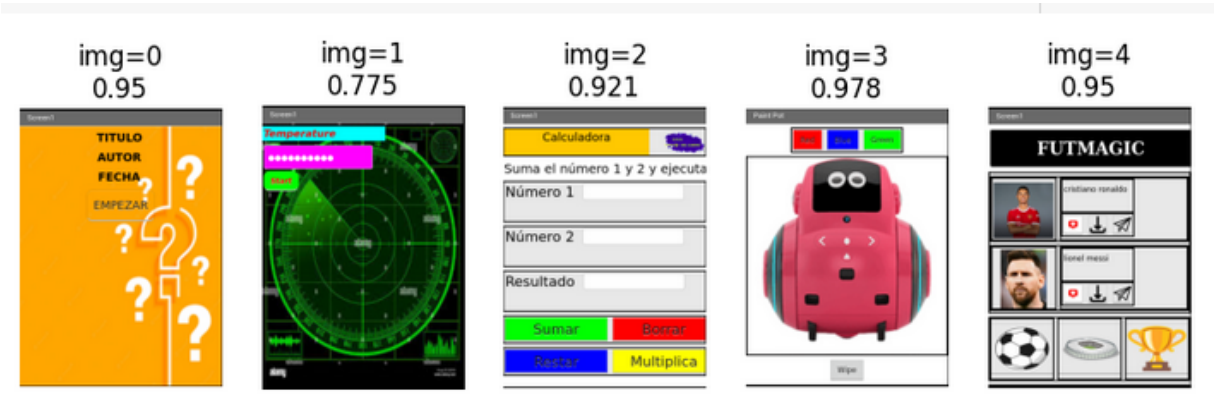
Ao final é avaliado o desempenho do modelo com base na diferença entre o *ranking* humano e o do modelo. A fim de avaliar a correlação entre as duas variáveis foi utilizado o coeficiente de Spearman.

4.5.1 Comparação com avaliação humana

Para avaliar o desempenho, foram comparados os resultados do modelo com avaliações humanas, definido originalmente por Kreuch (2021).

Tabela 15. Ranqueamento de aplicativos para teste do modelo.

app	**Cálculo automático**		***Ranking* por humanos 1 (mais similar) a 10 (menos similar)**
	Valor de similaridade calculado pelo modelo proposto	***ranking* baseado nos resultados do modelo**	
shopping_list	1.0	1	8
futmagic	0.9504	2	4
quiz	0.9501	3	3
paint_pot	0.9783	4	7
calculadora_1	0.9214	5	6
cooking	0.9123	6	1
número	0.8799	7	10
db	0.8586	8	5
calc_2	0.842	9	9
temperatura	0.7747	10	2



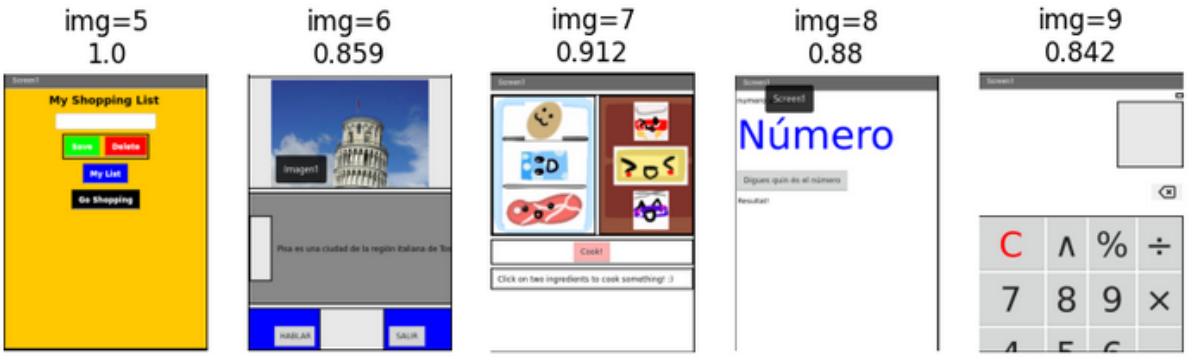


Figura 42. Capturas de tela de aplicativos retirados da galeria do App Inventor e suas respectivas notas atribuídas pelo modelo (Fonte: próprio autor).

Para avaliar a similaridade do modelo apresentado com as notas dadas por humanos, foi utilizado o coeficiente de similaridade de Spearman (1904), que mede a força e direção da associação de duas variáveis. O cálculo resulta num valor entre -1 e 1, sendo que quanto mais correlacionadas as variáveis, mais próximo de 1 resulta o cálculo. Como resultado da avaliação, o coeficiente de Spearman resultou no valor -0,03 que é uma correlação negativa e próxima de 0, indicando não haver correlação entre as variáveis.

4.5.2. Discussão

Primeiramente é necessário apontar que a abordagem teve um resultado negativo, o coeficiente de correlação entre as avaliações do modelo e avaliações humanas foi próximo de 0, indicando que, dentro deste contexto, o modelo não foi capaz de prever o ranqueamento de originalidade que um ser humano daria ao conjunto capturas de tela de aplicativos analisadas.

Foi também realizada uma segunda validação com um subconjunto dos aplicativos utilizados na avaliação do estudo de Kreuch (2021). Dentro deste contexto o modelo deste trabalho apresentou um resultado melhor, atingindo um coeficiente de Spearman de 0,25, apontando correlação entre as variáveis. Isso pode indicar que o conjunto de dados utilizados por Kreuch (2021) são mais representativos dos dados no conjunto de dados utilizado no treinamento.

É importante apontar, entretanto, que alguns desafios e dificuldades relacionados ao modelo *autoencoder* prejudicaram os resultados dessa pesquisa. A eficácia dos *autoencoders* depende de um conjunto de dados de treinamento amplo

e representativo. O conjunto de 8.010 telas do universo de referência provavelmente não foi suficiente para a tarefa de analisar a originalidade de telas de aplicativo proposta neste estudo.

Outro ponto a se acrescentar é a dificuldade de definir originalidade. A subjetividade do conceito o torna difícil estabelecer critérios claros e objetivos para rotular aplicativos como originais. Por consequência a tarefa de ranquear telas de aplicativos com base em sua originalidade efetuada pelo autor pode ter resultado em grandes divergências com o modelo, gerando resultados insatisfatórios.

Mais um dos fatores que prejudicaram os resultados deste trabalho foi a complexidade e diversidade dos *layouts* de aplicativos. *Layouts* são extremamente diversos e complexos, apresentando ampla variedade de elementos visuais, designs e estilos, isso levou a necessidade de utilizar um vetor complexo de 4.096 variáveis.

A abordagem do trabalho utilizou métodos não supervisionados para classificação de layouts. Nesse contexto, alguns pontos fracos dessa abordagem são a falta de interpretabilidade, pois métodos não supervisionados apesar de conseguirem detectar padrões, entender as razões por trás de suas escolhas é uma tarefa difícil, essa falta de interpretabilidade limita a explicação do cálculo de originalidade por trás do modelo. Outro ponto negativo é a subjetividade em determinar originalidade. Devido o aprendizado se dar somente a partir da estrutura dos dados, a concepção de originalidade de um método não supervisionado pode se desalinhar fortemente da concepção humana de originalidade

Alguns pontos fortes do método utilizado é não necessitar de anotação manual, isso facilita o processo de treinamento e permite aos pesquisadores utilizarem um conjunto muito maior de dados do que em métodos supervisionados, considerando limitações de força humana para o etiquetamento manual de dados.

Ameaças à validade. Este estudo sobre o uso de *autoencoders* para a avaliação da originalidade de aplicativos enfrenta ameaças à validade, em termos de validade interna e em relação ao tamanho da amostra.

A validade interna refere-se à confiabilidade e solidez dos resultados dentro do próprio estudo. No contexto do trabalho pode existir o viés de seleção, para mitigar esse risco os pesquisadores do GQS/INCoD/INE/UFSC coletaram as capturas de tela de maneira aleatória, bem como no código foi separado os conjuntos de treinamento e validação de forma automática e aleatória. Entretanto é

necessário apontar que há o risco de a amostra de dados do treinamento e validação não ser representativa, pois os dados de treinamento foram coletados em janeiro de 2021, enquanto *screenshots* utilizados na validação humana foram coletados em maio de 2023, momentos muito diferentes com tendências diferentes para aplicativos e seus designs, o que provavelmente contribuiu para a os resultados negativos deste estudo.

Existe também a possibilidade do viés de confirmação, entretanto foram utilizadas métricas objetivas para medir a eficácia do modelo, buscando uma abordagem o mais imparcial possível ao interpretar os resultados.

5. Conclusão

Como resultado do presente trabalho foi feita a fundamentação teórica sobre os principais conceitos e teoria referente a originalidade e computação utilizando *deep learning* (O1). Foi analisado também o estado da arte em relação a análise automática da originalidade de design de design de interfaces de apps (O2), constatando que atualmente as soluções para calcular a originalidade de telas de aplicativos são limitadas. Com base na fundamentação teórica e os resultados da revisão do estado da arte foi desenvolvida uma abordagem para automaticamente avaliar o design de interface de um app criado com App Inventor adotando uma abordagem não supervisionada diferente de trabalhos anteriores. Esse modelo utiliza um universo de referência de 8.010 imagens de *screenshots* de apps App Inventor, e por meio de uma representação calculada com base em *autoencoder*, busca reconhecer similaridades estruturais dos dados. A partir dessa representação é utilizada uma medida de similaridade comparando o design de um novo app com o universo de referência, calculado ao final uma nota de similaridade (O3). Em relação ao conjunto de testes composto por 10 apps, o modelo desenvolvido apresenta uma correlação negativa comparada com um ranking de originalidade dado por humanos, tendo em vista as razões apontadas na discussão do capítulo de avaliação do desempenho.

Considerando a natureza exploratória deste estudo sobre a avaliação de originalidade de aplicativos usando *autoencoders*, várias ameaças à validade foram identificadas. Uma dessas ameaças está relacionada à falta de uma definição mensurável e consensual do construto de originalidade. Embora exista um consenso

geral sobre a importância da originalidade em produtos criativos, a literatura não oferece uma definição precisa de como medir a originalidade. Para lidar com essa ameaça, a abordagem adotada neste estudo baseou-se em definições comumente encontradas na literatura para criatividade/originalidade de aplicativos.

Outra ameaça identificada está relacionada ao tamanho da amostra. Foi utilizada uma amostra menor que a de estudos relacionados. Logo, é importante ressaltar que permanece uma questão em aberto se os resultados obtidos neste estudo seriam melhores ao considerar um universo de aplicativos consideravelmente maior.

Apesar das ameaças à validade mencionadas, este estudo proporcionou entendimento preliminar sobre o uso de *autoencoders* na avaliação de originalidade de aplicativos. Os resultados obtidos, embora limitados ao escopo e às limitações do estudo, podem fornecer uma base para pesquisas futuras e estimular discussões sobre a medição e a detecção de originalidade em layouts de aplicativos. Ainda há espaço para aprimoramentos metodológicos e expansão do estudo para uma amostra mais representativa, com o objetivo de validar e generalizar os resultados encontrados. Em suma, este trabalho teve como intuito contribuir na investigação da avaliação de originalidade de aplicativos por meio de *autoencoders*. As ameaças à validade identificadas demonstram as limitações e desafios encontrados, destacando a necessidade de pesquisas adicionais nessa área para melhorar a compreensão e a aplicabilidade dessa abordagem na prática.

REFERÊNCIAS

ALVES, N. C., WANGENHEIM, C. G., ALBERTO, M., HELENA L. M. P. Uma Proposta de Avaliação da Originalidade do Produto no Ensino de Algoritmos e Programação na Educação Básica. Anais do Simpósio Brasileiro de Informática na Educação, Natal, Brasil, 2020.

BELTRAMELLI, T. pix2code: Generating Code from a Graphical User Interface Screenshot, the ACM SIGCHI Symposium, Copenhagen, Denmark, 2017.

BESEMER, S., TREFFINGER, D. J. Analysis of Creative Products: Review and Synthesis. Journal of Creative Behavior, 15(3), 158-178 1981.

BISHOP, C.. Pattern recognition and machine learning. Springer, Berlin, 2006.

BROWNLEE, J., 2020. How do Convolutional Layers Work in Deep Learning Neural Networks? **https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/**

Acesso em 22/07/2022

BUNIAN, S., LI, K., JEMMALI, C., HARTVELD, C., FU, Y., SEIF, M. E. Vins: Visual search for mobile user interface design**.** Proc. of the CHI Conference on Human Factors in Computing Systems. Yokohama, Japan, 2021.

CAVALLO, D., GOMES, A. S., SINGER, H., IBERT, I. B. Inovação e Criatividade na Educação Básica: Dos conceitos ao ecossistema. Revista Brasileira de Informática na Educação, 24(2), 2016.

CHEN, J., CHEN, C., XING, Z., XIA, X., ZHU, L., GRUNDY, J., WANG, J. Wireframe-based UI Design Search through Image Autoencoder. ACM Transactions on Software Engineering and Methodology, 29(3), Article 19, 2020.

CHOI, R. Y., AARON, S., KALPATHY, J., C., MICHAEL, F. C., CAMPBELL, P. J. Introduction to Machine Learning, Neural Networks, and Deep Learning. TVST, 9(2), 2022.

CRASWELL, N. Mean Reciprocal Rank. In: LIU, L., ÖZSU, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA, 2009.

FEIZ, S., WU, J., ZHANG, X., SWEARNGIN, A., BARIK, T., NICHOLS, J. Understanding Screen Relationships from Screenshots of Smartphone Applications. Proc. of the 27th International Conference on Intelligent User Interfaces, Helsinki, Finland, 2022.

FERREIRA, M. N. F., PINHEIRO, F. da C., GRESSE VON WANGENHEIM, C., FILHO, R. M., HAUCK, J. C. R. Ensinando design de interface de usuário de aplicativos móveis no ensino fundamental. Revista Brasileira de Informática na Educação, 28(0), 2020.

DEKA et al., B. et al. Rico: A Mobilie App dataset for Building Data-Driven Design Applications. Proc. of the 30th Annual ACM Symposium on User Interface Software and Technology, 20 out. 2017.

DONALEK, C. Supervised and Unsupervised Learning. Caltech.edu, 2011.

GARRET, J. The elements of user experience. New Riders, 2º edition, 2011.

GE, X. Android GUI search using hand-drawn sketches. Proc. of the IEEE/ACM 41st International Conference on Software Engineering, Montreal, Canada, 2019.

GROVER, S.; BASU, S.; SCHANK, P. What we can learn about student learning from open-ended programming projects in middle school computer science. Proc. of the 49th ACMTechnical Symposium on Computer Science Education. Baltimore, EUA, 2018

GUILFORD. J.P., Deinin. Creativity. American Psychologist, *5*(9), 1950.

HAYKIN, S. Redes Neurais: Princípios e Prática, 2ª edição. Porto Alegre, RS, Brasil, Bookman Editora, 2007.

HUANG, F; CANNY, J. F.; NICHOLS, J. Swire: Sketch-based user interface retrieval. Proc. of the CHI Conference on Human Factors in Computing Systems. Glasgow, Scotland, 2019.

IBM Cloud Learn, 2020. O que é machine learning?

https://www.ibm.com/br-pt/cloud/learn/machine-learning. Acesso em: 22/07/2022

IREKPONOR, V. E. Mathematical Prerequisites For Understanding Autoencoders and Variational Autoencoders (VAEs): Beginner Friendly, Intermediate Exciting, and Expert Refreshing.

https://medium.com/analytics-vidhya/mathematical-prerequisites-for-understanding-autoencoders-and-variational-autoencoders-vaes-8f854025390e. Acesso em: novembro 2022

JORDAN, J., 2018. Introduction to Autoencoders

https://www.jeremyjordan.me/autoencoders/ Acesso em: dezembro 2022

KREUCH, L. Desenvolvimento de um modelo de avaliação da originalidade de design de interface de aplicativos Android. Trabalho de Conclusão de Curso. (Graduação em Ciências da Computação) Universidade Federal de Santa Catarina. Florianópolis, 2022.

LECUN, Y, BENGIO, Y., HINTON, G. Deep Learning. 444, NATURE, 521, 2015.

LI, T. J., POPOWSKI, L., MITCHELL, T. M., MYERS, B. A. Screen2Vec: Semantic Embedding of GUI Screens and GUI Components. arXiv:2101.11103, 2021.

LIU, T. F., CRAFT, M., SITY, J., YUMER, E., MECH, R., KUMAR, R. Learning Design Semantics for Mobile Apps.Proc. of the 31st Annual ACM Symposium on User Interface Software and Technology Berlin, Germany. 2018.

LOGUNOVA, I. 2022. Deep Learning Applications for Computer Vision.

https://serokell.io/blog/deep-learning-for-computer-vision. Acesso em: setembro 2022.

LYE, S., KOH, J. H. L. Review on teaching and learning of computational thinking through programming: What is next for K-12? Computers in Human Behavior, 41, 51-61, 2014.
FERREIRA, M. V. R., WANGENHEIM, C. G. ALVES, N. C.. Desenvolvimento de um Modelo de Avaliação da Originalidade de Aplicativos Móveis Usando Técnicas de *Machine Learning*. Relatorio tecnico, INCoD-GQS.064.2021P. 2021, INCoD/INE/UFSC, Florianopolis, Brasil, 2021..

MASSI, M., 2019. Schema of a basic Autoencoder.
https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_schema.png Acesso em: 22/07/2022

NGUYEN, T. T., MINH, P. V., PHAM, H. V., NGUYEN, T. T. Deep Learning UI Design Patterns of Mobile Apps. Proc. of ACM/IEEE 40th International Conference on Software Engineering: New Ideas and Emerging Results, Gothenburg, Sweden, 2018.

O'SHEA, K. An Introduction to Convolutional Neural Networks, Aberystwyth University, Wales, UK, 2015.

RUNCO, M. A., JAEGER, G. J. The standard definition of creativity. Creativity Research Journal, 24(1), 92-96, 2012.

PAPANEK, V. Design for the real world. New York. Pantheon Books, 1971

RHODES, M. An analysis of creativity. Phi Delta Kappan, 42, 305–310, 1961.

SINGH, S. G. KNN Vs. K-Means. https://www.codingninjas.com/codestudio/library/knn-vs-k-means. Acesso em novembro 2022.

SOUZA, A. S. Uma abordagem para avaliação de originalidade de design de interface de usuário de aplicativos móveis utilizando técnicas de inteligência artificial para a educação básica. Universidade Federal de Santa Catarina, Florianópolis, 2022.

SOUZA, A. S., ALVES, N. C., WANGENHEIM, C. G., KREUCH, L. Análise Automatizada da Originalidade de Design de Interfaces de Usuário no Contexto Educacional: Um Mapeamento da Literatura. Anais do Simpósio Brasileiro de Informática na Educação, Porto Alegre, Brasil, 2021.

SUN, M., LI, M., LUI, J. C. S. Droid Eagle: seamless detection of visually similar Android apps, Department of Computer Science and Engineering, The Chinese University of Hong Kong, 2015.

TAN, S., TIAN, Z., ZHONG, X., YU, S. A Novel Android Malware Detection Method Based on Visible User Interface. Proc. IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications, Naples, Italy, 2021.

VANDERPLAS, J. Python Data Science Handbook. O'Reilly Media, Inc. 2016.

WANGENHEIM, C. G, WANGENHEIM, A.. Overview on a human-centric interactive ML process for teaching ML in K-12. Working paper WP_GQS_01_2021_v10, GQS/INCoD/UFSC, 2021.

WU, Z., 2019. Understanding and Modeling User-Perceived Brand Personality from Mobile Application UIs. Proc. of the CHI Conference on Human Factors in Computing Systems. Glasgow, Scotland, UK, 2019

XIE, Y.; LIN, T.; XU, H. User Interface Code Retrieval: A Novel Visual-Representation-Aware Approach. IEEE access, 7 , 2019.

YOO, H. J. Deep Convolution neural Networks in Computer Vision: a Review. IEIE Transactions on Smart Processing and Computing, 4(1), 2015.

YU, S., et al. Layout and Image Recognition Driving Cross-Platform Automated Mobile Testing. Proc. of the IEEE/ACM 43rd International Conference on Software Engineering, Madrid, Spain, 2021.

APÊNDICE A

Para realizar o treinamento da rede utilizando o Google Colab foi utilizado o seguinte código:

```
# For commands
import os
#os.chdir('/content/')
import time
from tqdm.notebook import tqdm
import warnings
warnings.filterwarnings('ignore')
# For array manipulation
import numpy as np
import pandas as pd
import pandas.util.testing as tm
# For visualization
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
import cv2
import imageio as io
from pylab import *
from sklearn.manifold import TSNE
#For model performance
from sklearn.metrics import pairwise_distances
```

```
from sklearn.metrics.pairwise import cosine_similarity
#from sklearn.externals import joblib
#For model training
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import calinski_harabasz_score,
silhouette_score, davies_bouldin_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from keras.applications.vgg16 import preprocess_input
from tensorflow.keras.layers import Conv2D, Activation,
MaxPooling2D, UpSampling2D
from tensorflow.keras.optimizers import Adam, Adagrad, RMSprop
from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint
from keras import backend as K
from keras.models import load_model
from keras.preprocessing import image
from matplotlib.colors import ListedColormap
import joblib
from scipy.spatial import distance
from sklearn import preprocessing

from google.colab import drive
drive.mount('/content/drive')

rootdir = '/content/drive/MyDrive/TCC
klaus/Screenshots_App_Inventor/Screenshots_App_Inventor'

file_path = []
rootdir = '/content/drive/MyDrive/TCC
klaus/Screenshots_App_Inventor/Screenshots_App_Inventor'
for root, subdirs, files in os.walk(rootdir):
  for filename in files:
     curr_path = os.path.join(root, filename)
     if " (1)" in curr_path and curr_path.endswith(".png"):
```

```
        file_path.append(curr_path)
        # print(curr_path)


print("Images: ", len(file_path))

train_files, test_files = train_test_split(file_path, test_size =
0.15)
print(len(train_files))
print(len(test_files))



train_files = pd.DataFrame(train_files, columns=['filepath'])
test_files = pd.DataFrame(test_files, columns=['filepath'])
#converting into .csv file for future reference.
train_files.to_csv(f'{rootdir}/train_file.csv')
test_files.to_csv(f'{rootdir}/test_file.csv')

#loading csv files.
train_files                                                    =
list(pd.read_csv(f'{rootdir}/train_file.csv')['filepath'])
test_files                                                     =
list(pd.read_csv(f'{rootdir}/test_file.csv')['filepath'])

        image_array = []


def image2array(file_array):
        image_array = []
        for path in tqdm(file_array):
        img = cv2.imread(path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (64,64))
        image_array.append(np.array(img))
        image_array = np.array(image_array)
        image_array  =  image_array.reshape(image_array.shape[0],  64,
64, 3)
```

```
    image_array = image_array.astype('float32')
    image_array /= 255
    return np.array(image_array)



train_data = image2array(train_files)
print("Length of training dataset:",train_data.shape)
test_data = image2array(test_files)
print("Length of test dataset:",test_data.shape)

## save the np array
with  open(f'{rootdir}/converted_images_to_array_train.npy',  'wb')
as f:
    np.save(f, train_data)
with open(f'{rootdir}/converted_images_to_array_test.npy', 'wb') as
f:
    np.save(f, test_data)

# load array back
train_data = None
test_data = None
with  open(f'{rootdir}/converted_images_to_array_train.npy',  'rb')
as f:
    train_data = np.load(f)
with open(f'{rootdir}/converted_images_to_array_test.npy', 'rb') as
f:
    test_data = np.load(f)

def encoder_decoder_model():


    """
    #Encoder
    model = Sequential(name='Convolutional_AutoEncoder_Model')
```

```
    model.add(Conv2D(8,                              kernel_size=(3,
3),activation='relu',input_shape=(64,    64,    3),padding='same',
name='Encoding_Conv2D_1'))
    model.add(MaxPooling2D(pool_size=(2,      2),      strides=2,
padding='same', name='Encoding_MaxPooling2D_1'))
    model.add(Conv2D(16,                             kernel_size=(3,
3),strides=1,kernel_regularizer                                  =
tf.keras.regularizers.L2(0.001),activation='relu',padding='same',
name='Encoding_Conv2D_2'))
    model.add(MaxPooling2D(pool_size=(2,      2),      strides=2,
padding='same', name='Encoding_MaxPooling2D_2'))



    model.add(Conv2D(16,  kernel_size=(3,  3),  activation='relu',
kernel_regularizer       =       tf.keras.regularizers.L2(0.001),
padding='same',name='Decoding_Conv2D_4'))
    model.add(UpSampling2D((2, 2),name='Decoding_Upsamping2D_4'))
    model.add(Conv2D(8,  kernel_size=(3,  3),  activation='relu',
kernel_regularizer       =       tf.keras.regularizers.L2(0.001),
padding='same',name='Decoding_Conv2D_5'))
    model.add(UpSampling2D((2, 2),name='Decoding_Upsamping2D_5'))
    model.add(Conv2D(3,            kernel_size=(3,            3),
padding='same',activation='sigmoid',name='Decoding_Output'))
    return model

model = encoder_decoder_model()
model.summary()
print("\n")
tf.keras.utils.plot_model(model,       to_file='model_kmeans.png',
show_shapes=True)

optimizer = Adam(learning_rate=0.001)
model = encoder_decoder_model()
m = []
model.compile(optimizer=optimizer,                      loss='mse',
metrics=[tf.keras.metrics.Accuracy(),
```

```
tf.keras.metrics.MeanSquaredError(),
tf.keras.metrics.KLDivergence()])
early_stopping = EarlyStopping(monitor='val_loss',
mode='min',verbose=1,patience=6,min_delta=0.0001)
checkpoint = ModelCheckpoint('content/drive/MyDrive/TCC
klaus/encoder_model_checkpoint.h5', monitor='val_loss', mode='min',
save_best_only=True)
model.fit(train_data, train_data, epochs=35,
batch_size=8,validation_data=(test_data,test_data),callbacks=[early
_stopping,checkpoint])

# save model
model.save(f'{rootdir}/encoder_model')

# load model
optimizer = Adam(learning_rate=0.001)
model = load_model(f'{rootdir}/encoder_model')
model.compile(optimizer=optimizer, loss='mse')

from keras import backend as K
def feature_extraction(model, data, layer = 4):

    """
    Creating a function to run the initial layers of the
encoder model. (to get feature extraction from any layer of the
model)
    Arguments:
    model - (Auto encoder model) - Trained model
    data - (np.ndarray) - list of images to get feature
extraction from trained model
    layer - (int) - from which layer to take the features(by
default = 4)
    Returns:
    pooled_array - (np.ndarray) - array of extracted features
of given images
```

```
    """

    encoded                                                    =
K.function([model.layers[0].input],[model.layers[layer].output]
)
    encoded_array = encoded([data])[0]
    pooled_array = encoded_array.max(axis=-1)
    return encoded_array

def get_batches(data, batch_size=1000):

    """
    Taking batch of images for extraction of images.
    Arguments:
    data - (np.ndarray or list) - list of image array to get
extracted features.
    batch_size - (int) - Number of images per each batch
    Returns:
    list - extracted features of each images
    """

    if len(data) < batch_size:
    return [data]
    n_batches = len(data) // batch_size

    # If batches fit exactly into the size of df.
    if len(data) % batch_size == 0:
    return  [data[i*batch_size:(i+1)*batch_size]  for  i  in
range(n_batches)]

    # If there is a remainder.
    else:
    return            [data[i*batch_size:min((i+1)*batch_size,
len(data))] for i in range(n_batches+1)]
```

```
d = np.concatenate([train_data,test_data],axis=0)

X_encoded = []
i=0
# Iterate through the full training set.
for batch in get_batches(d, batch_size=300):
     i+=1
     # This line runs our pooling function on the model for
each batch.
     X_encoded.append(feature_extraction(model, batch))

X_encoded = np.concatenate(X_encoded)

X_encoded_double = X_encoded.astype('double')
X_encoded_reshape_double                                    =
X_encoded_double.reshape(X_encoded.shape[0],
X_encoded.shape[1]*X_encoded.shape[2]*X_encoded.shape[3])
print('Encoded shape:', X_encoded_reshape_double.shape)

with  open(f'{rootdir}/X_encoded_compressed_double.npy',  'wb')
as f:
     np.save(f, X_encoded_reshape_double)

# load double version
X_encoded_reshape_double = None
with  open(f'{rootdir}/X_encoded_compressed_double.npy',  'rb')
as f:
     X_encoded_reshape_double = np.load(f)
print(X_encoded_reshape_double.shape)

def
plot_(x,y1,y2,row,col,ind,title,xlabel,ylabel,label,isimage=Fal
se,color='b'):
     plt.subplot(row,col,ind)
```

```
     if isimage:
     plt.imshow(x)
     plt.title(title)
     plt.axis('off')
     else:
     plt.plot(y1,label=label,color='g');
plt.scatter(x,y1,color='g')
     if   y2!='':  plt.plot(y2,color=color,label='validation');
plt.scatter(x,y2,color=color)
     plt.grid()
     plt.legend()
     plt.title(title); plt.xlabel(xlabel); plt.ylabel(ylabel)


lisp=train_files
lisp.extend(test_files)
print(len(lisp))


K = [7]

calinski_harabasz_score_list = []
silhouette_score_list = []
davies_bouldin_score_list = []

file_list = train_files
kmeans = None
for k in K:
     print("if Number of clusters: "+str(k))
     kmeans = KMeans(n_clusters = k,
random_state=0).fit(X_encoded_reshape)
     labels=kmeans.labels_

     # measuring how good the clusters are

calinski_harabasz_score_list.append(calinski_harabasz_score(X_encod
ed_reshape, labels))

silhouette_score_list.append(silhouette_score(X_encoded_reshape,
labels))

davies_bouldin_score_list.append(davies_bouldin_score(X_encoded_res
hape, labels))
```

```
      centroids = kmeans.cluster_centers_
      plt.figure(figsize=(10,5))
      plt.subplot(1,1,1)
      plt.scatter(values[:,0], values[:,1], c=
kmeans.labels_.astype(float), s=50, alpha=0.5)
      plt.scatter(centroids[:, 0], centroids[:, 1], c=None, s=50)
      plt.show()
      print(values[:,0], centroids[:,0])

      for group in range(k):
      iter=0
      plt.figure(figsize=(13,3))
      for i in range(len(labels)):
            label = labels[i]
            if label == group:
                  img = cv2.imread(file_list[i])

plot_(img,"","",1,6,iter+1,"cluster="+str(group)+"\nimg="+str(i)+"\
nfile="+str(file_list[i]),"","","",True)
                  iter+=1
            if iter>=5: break
      plt.show()
      print()

best_number_of_clusters = 7
kmeans = KMeans(n_clusters = 7,
random_state=0).fit(X=X_encoded_reshape.astype('double'))
labels=kmeans.labels_
centroids = kmeans.cluster_centers_
cluster_size_map = {}
for i in range(best_number_of_clusters):
  cluster_size_map[i] = 0
for label in kmeans.labels_:
  cluster_size_map[label] += 1

clusters_features = []
cluster_files=[]
for i in [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]:
      i_cluster = []
      i_labels=[]
      for iter,j in enumerate(kmeans.labels_):
      if j==i:
            i_cluster.append(X_encoded_reshape[iter])
            i_labels.append(lisp[iter])
      i_cluster = np.array(i_cluster)
      clusters_features.append(i_cluster)
      cluster_files.append(i_labels)
```

```
labels=[]
data=[]
files=[]
for iter,i in enumerate(clusters_features):
     data.extend(i)
     print(i)
     labels.extend([iter for i in range(i.shape[0])])
     files.extend(cluster_files[iter])

def get_medium_distance_from_cluster_centers(sample, cluster):
  sum_of_distances = 0
  for i in range(best_number_of_clusters):
     # print('centroid distance', distance.euclidean(sample,
centroids[i]))
     if i == cluster:
     continue
     sum_of_distances += distance.euclidean(sample, centroids[i])
  return sum_of_distances / (best_number_of_clusters - 1)


# new prediction
def evaluate_originality(screenshot, cluster=None, debug=False):
  if cluster == None:
     cluster = kmeans.predict([screenshot])

  cluster_center = centroids[cluster]
  distance_intra_cluster = distance.euclidean(screenshot,
cluster_center[0])

  mean_distance_from_all_clusters =
get_medium_distance_from_cluster_centers(screenshot, cluster)

  number_of_screenshots_in_cluster = cluster_size_map[cluster[0]]

  prediction = distance_intra_cluster +
mean_distance_from_all_clusters -
number_of_screenshots_in_cluster/100

  if debug:
     print(f'prediction: {prediction}, is on cluster {cluster[0]},
distance_intra_cluster[{distance_intra_cluster}] +
mean_distance_from_all_clusters[{mean_distance_from_all_clusters}]
-
number_of_screenshots_in_cluster[{number_of_screenshots_in_cluster}
')

  return prediction
```

```
originality_scores = []
worst = []
medium = []
best = []
for iter, d in enumerate(data):
  a = d.tolist()
  prediction = evaluate_originality(a, debug=False)
  originality_scores.append(prediction)

# normalizing using simple code\
def normalize_list():
     minimum = min(originality_scores)
     maximum = max(originality_scores)
     normalized = [(x - minimum) / (maximum - minimum) for x in
originality_scores]
     return normalized

def normalize_values(list_scores, new_values):
     extended_list = list_scores + new_values
     minimum = min(extended_list)
     maximum = max(extended_list)
     normalized_new_values = []
     for value in new_values:
     normalized = (value - minimum) / (maximum - minimum)
     normalized_new_values.append(normalized)
     return normalized_new_values

X_normalized = normalize_list()
print(len(X_normalized))
count, bins = np.histogram(np.array(X_normalized))
plt.stairs(count, bins)

X_normalized = preprocessing.normalize([originality_scores])
print(len(X_normalized[0]))
count, bins = np.histogram(X_normalized)
plt.stairs(count, bins)

# compare Kreuch's 10 apps using my model

kreuch_file_paths = []
images_kreuch = '/content/drive/MyDrive/TCC
klaus/Screenshots_App_Inventor/avaliacao_humana_2'
for root, subdirs, files in os.walk(images_kreuch):
  for filename in files:
     curr_path = os.path.join(root, filename)
     kreuch_file_paths.append(curr_path)
     print(curr_path)
```

```
kreuch_image_array = image2array(kreuch_file_paths)

encoded_image_array = feature_extraction(model, kreuch_image_array)

non_normalized_scores = []
for encoded_image in encoded_image_array:

  flattened_encoded_image =
np.array(encoded_image).flatten().reshape(1,-1)

  # evaluation
  encoded_array = flattened_encoded_image.tolist()
  prediction = evaluate_originality(encoded_array[0], debug=False)
  non_normalized_scores.append(prediction)
print(non_normalized_scores)
normalized_new_values = normalize_values(originality_scores,
non_normalized_scores)

plt.figure(figsize=(20,5))
for iter,i in enumerate(kreuch_file_paths):

plot_(read(i),"","",1,10,iter+1,'img='+str(iter)+'\n'+str(round(nor
malized_new_values[iter], 4)),"","","",True)
plt.show()
```

APÊNDICE B

Neste apêndice é apresentado o artigo no formato SBC, referente ao presente projeto

Desenvolvimento de um Modelo de Avaliação da Originalidade de Design de Interface de Aplicativos Android

Klaus de Freitas Dornsbach

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

Abstract. *In the rapidly evolving 21st century, the demand for acquiring new skills has become indispensable in both daily life and the job market. Notably, creativity has emerged as a vital skill, encompassing the ability to generate fresh and practical ideas. To cultivate creativity in Basic Education, a promising approach involves integrating block programming environments like App Inventor. By teaching app development, this method not only imparts programming knowledge but also emphasizes the significance of interface design. Consequently, it becomes crucial to evaluate the originality of students' application layouts as part of the teaching-learning process. In light of this, the aim of this project is to develop an automated model utilizing non supervised deep learning techniques, in order to assess the originality of interface designs in applications developed using App Inventor in the context of teaching computer science in Basic Education.*

Resumo. *No século XXI em rápida evolução, a procura de novas competências tornou-se indispensável tanto na vida quotidiana como no mercado de trabalho. Notavelmente, a criatividade emergiu como uma habilidade vital, abrangendo a capacidade de gerar ideias novas e práticas. Para cultivar a criatividade na Educação Básica, uma abordagem promissora envolve a integração de ambientes de programação de blocos como o App Inventor. Ao ensinar o desenvolvimento de aplicativos, esse método não apenas transmite conhecimento de programação, mas também enfatiza a importância do design de interface. Consequentemente, torna-se*

crucial avaliar a originalidade dos layouts de aplicativos dos alunos como parte do processo de ensino-aprendizagem. Diante disso, o objetivo deste projeto é desenvolver um modelo automatizado utilizando técnicas de aprendizado profundo não supervisionado, a fim de avaliar a originalidade de designs de interface em aplicativos desenvolvidos usando o App Inventor no contexto do ensino de informática na Educação Básica.

1. Introdução

A criatividade é uma habilidade essencial no século XXI [Cavallo et al., 2016], logo o desenvolvimento dessa habilidade deve fazer parte da formação do indivíduo desde cedo. Entre outras disciplinas ela também pode ser desenvolvida pelo ensino de computação [da Cruz Alves et al., 2020]. Uma forma de ensinar computação na educação básica pode ser por meio de desenvolvimento de apps com App Inventor (appinventor.mit.edu), incluindo o projeto de design de interface como também a programação do parte funcional do app [Ferreira et al., 2020].

Design de interface de usuário é o processo pelo qual designers constroem interfaces, ou portas para interação, para usuários de software ou dispositivos computadorizados. Um design de interface de aplicativos engloba o *layout,* que é a forma em que elementos ficam dispostos numa página.

No contexto do ensino de computação, uma das estratégias mais usadas é a metodologia ativa usando o ciclo "Use-Modifique-Crie" (UMC) [da Cruz Alves, 2020]. Seguindo esse ciclo o aluno aprende conceitos importantes analisando de um programa pronto, para que posteriormente possa criar um artefato próprio, demonstrando também sua criatividade por meio de um *layout* de Interface de usuário (UI). Nessas circunstâncias, é importante uma avaliação e *feedback* acerca da aprendizagem da criatividade, ajudando o estudante no seu processo de aprendizagem. Entretanto, avaliações manuais feitas por um ser humano estão sujeitas a serem subjetivas e trabalhosas, uma alternativa para mitigar o risco de avaliações injustas e poupar tempo dos professores de computação é a automatização desse processo [da Cruz Alves, 2020].

Para criar um avaliador automático é necessário uma definição de criatividade do produto. Mesmo sendo um conceito complexo, existem algumas definições que revolvem ao redor da ideia de criatividade incluir utilidade, adequação e a originalidade [Besemer and

Treffinger, 1981]. Consequentemente é necessário que uma avaliação de criatividade contemple a dimensão da originalidade. Porém a maioria das abordagens de avaliação automática existentes no contexto do ensino de computação avaliam originalidade por meio da divergência entre códigos fonte, não abordando características visuais como design de interface, nem características funcionais [da Cruz Alves et al., 2020].

Assim, a proposta deste trabalho de conclusão de curso é criar um modelo que automaticamente avalie o design de interface criado pelo estudante em relação a originalidade do *layout* de UI no contexto de aplicativos criados no App Inventor. Já existem abordagens, inclusive pesquisas realizadas por pesquisadores da iniciativa Computação na Escola [Souza et al., 2021] que utilizam abordagens supervisionadas para identificar componentes do layout de um aplicativo e medir similaridade com base nisso, calculando uma nota de originalidade. Kreuch utilizou um subconjunto de aproximadamente 4% do conjunto de dados para o treino do módulo que realiza reconhecimento de componentes de interface, abaixo da média de 70% nos estudos avaliados na pesquisa bibliográfica,

Diferente destas abordagens usando aprendizagem supervisionada, o presente trabalho visa utilizar métodos não supervisionados para aumentar o tamanho do conjunto de dados de treino por meio de treinamento não supervisionado.

Portanto, foi realizada uma pesquisa acerca da originalidade no design de *layouts* e visa-se a adoção de técnicas de *deep learning*, treinando um modelo para avaliar automaticamente a originalidade de *layouts,* procurando utilizar métodos diferentes dos já explorados, de maneira a compará-los.

2. Originalidade no contexto de interface de apps

Um produto pode ser definido como sendo um objeto físico, um artefato computacional, como p.ex. um app, que não está unicamente ligado com a vida de um indivíduo [Brogden and Sprecher, 1964]. A definição de criatividade focada no produto geralmente inclui características como novidade (quão incomum ou original o produto é), adequação (se o produto serve ao seu propósito e este propósito é útil) e condensação (quão atrativo o produto é) [Besemer and Treffinger, 1981]. Entre essas características, o presente trabalho foca especificamente na característica de novidade. A novidade de um aplicativo modelo pode se relacionar a diversos planos, especificamente na originalidade do design de interface.

O App Inventor (appinventor.mit.edu) permite a criação de aplicativos por meio de interface de usuário [Patton et al., 2019]. A parte gráfica de interface de usuário é definida arrastando e posicionando blocos com elementos visuais, e na parte de programação destes elementos, são disponibilizados blocos que se encaixam e formam sequências de comandos. Existe uma distinção entre um ambiente de design de interface, e de programação em que sequências de instruções são definidas. Devido à sua simplicidade, ambientes de programação baseado em bloco como o App Inventor são utilizados no ensino de computação (MIT, 2021)

3. Modelo de avaliação automática de originalidade da tela de aplicativos desenvolvidos com App Inventor

O modelo recebe como entrada o *screenshot* da tela de um aplicativo Android, no contexto educacional. Esta captura de tela é tirada de um aplicativo novo criado por um estudante como resultado de seu aprendizado de criação de aplicativos no App Inventor. O modelo avalia a originalidade da tela comparando com um universo de referência. Este universo de referência é um conjunto de imagens utilizadas no treinamento do modelo.

Foi utilizado o modelo de *autoencoder* para resolver a questão do reconhecimento e sumarização de *features*. Foi encontrado amplo respaldo em pesquisas utilizando *autoencoder* para resolver problemas de classificação de similaridade de aplicativos. Os *autoencoders* são compostos de diversas camadas de convolução e *max pooling* de forma a criar uma representação simplificada de uma entrada, e em seguida reconstrução por meio de camadas de convolução e *up sampling*, medindo o erro entre a entrada e saída para atualização dos pesos.

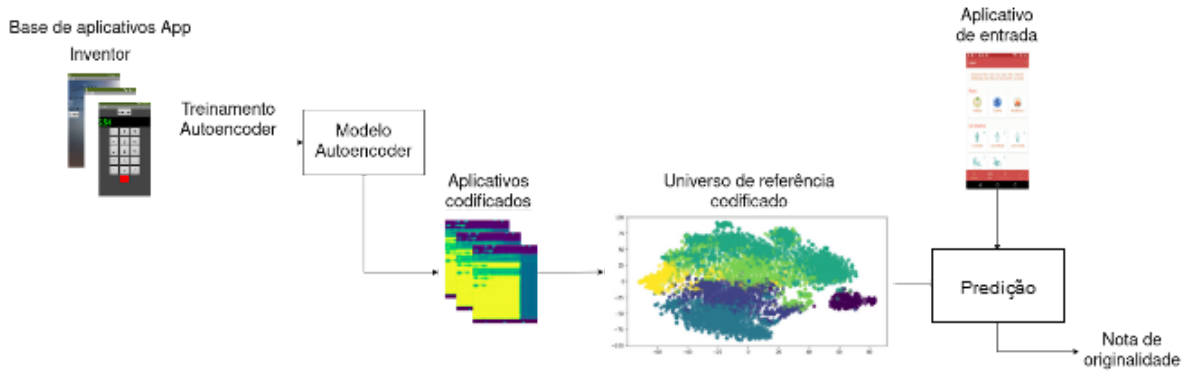


Figura 1. Modelo idealizado (Fonte: próprio autor).

3.1 Conjunto de dados

Levando em conta o foco em aplicativos desenvolvidos por meio de App Inventor, são utilizados os *screenshots* coletados por pesquisadores do GQS/INCoD/INE/UFSC em Janeiro de 2021. Esses 8.010 *screenshots* foram coletados de 1.551 apps selecionados aleatoriamente de um conjunto de mais de 88 mil apps da galeria do App inventor fornecido pelo MIT. O conjunto de dados coletado possui uma resolução de imagens de 1.080x1.920 pixels. A Figura 2 apresenta exemplos de *screenshots* do conjunto de dados.

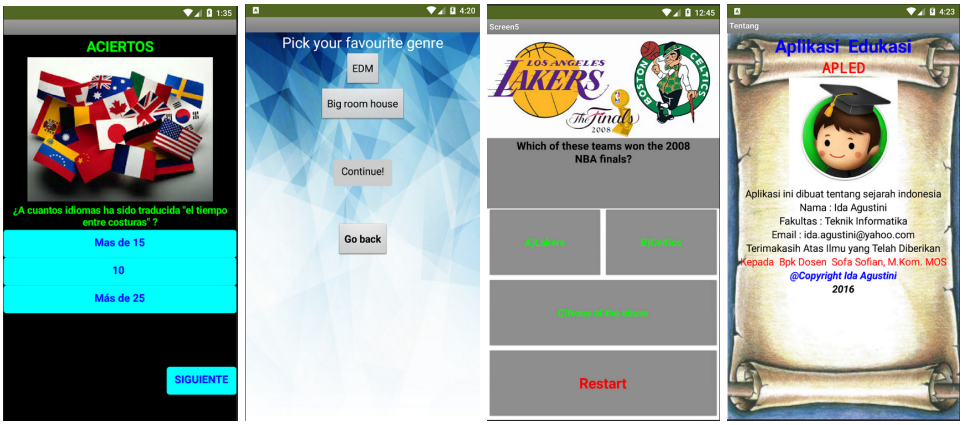


Figura 2. Exemplos de *screenshots* do *dataset* (Fonte: INCoD).

3.2 Treinamento do modelo de ML

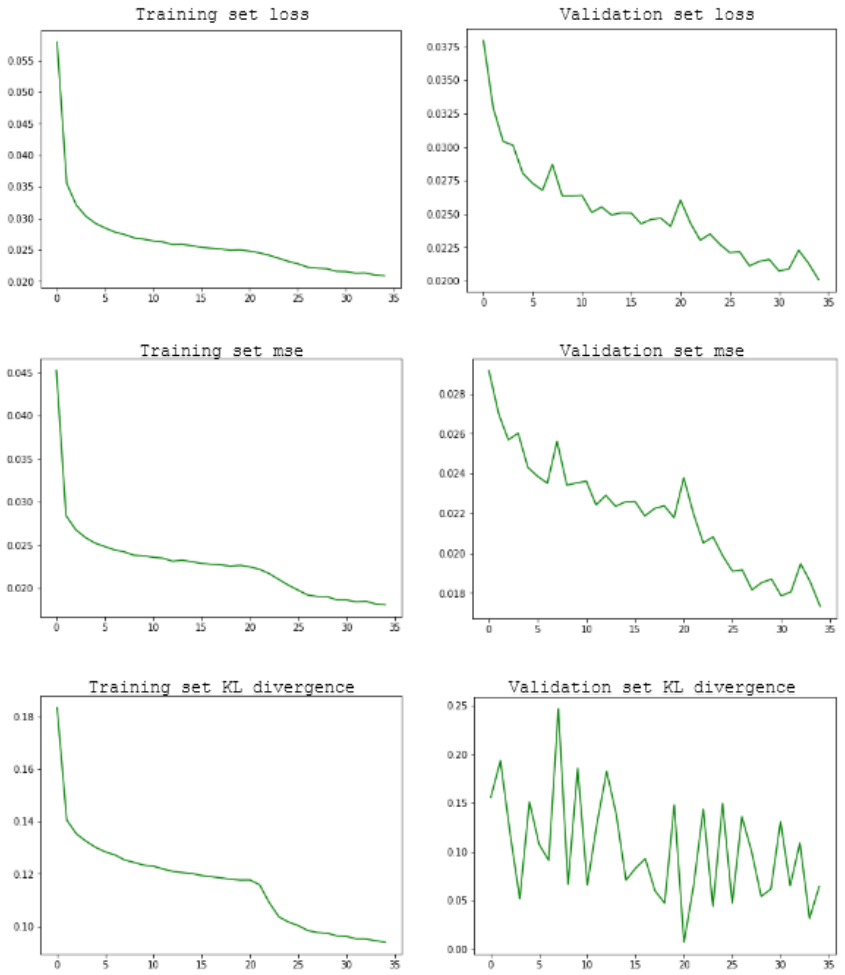
Seleção do Modelo. Foi utilizado o modelo de *autoencoder* para resolver a questão do reconhecimento e sumarização de *features*. Foi encontrado amplo respaldo em pesquisas utilizando *autoencoder* para resolver problemas de classificação de similaridade de aplicativos. Os *autoencoders* são compostos de diversas camadas de convolução e *max pooling* de forma a criar uma representação simplificada de uma entrada, e em seguida reconstrução por meio de camadas de convolução e *up sampling*, medindo o erro entre a entrada e saída para atualização dos pesos.

Foi utilizada a biblioteca Keras do python, que por sua vez emprega Tensor Flow, escrito em python e C++ para alto desempenho, em específico multiplicação de matrizes. O modelo utiliza o algoritmo de Adam [Kingma et al., 2015], que é uma extensão do algoritmo de gradiente estocástico descendente. A maior diferença entre estes está no algoritmo clássico utilizar um taxa de aprendizagem fixo para todos os pesos durante todo o treinamento,

enquanto Adam utiliza taxa de aprendizagem com valores diferentes para cada peso e que seguem uma média móvel exponencial, dando ênfase nas alterações mais recentes no gradiente. Para este algoritmo, uma taxa de aprendizado comumente utilizada é 0.001 [Kingma et al., 2015], nos testes do modelo foi sempre utilizado este número como taxa inicial. Após uma série de experimentos, foram utilizados estes parâmetros para o treinamento do modelo conforme especificado na Tabela 1.

Tabela 1. Parâmetros finais utilizados no treinamento do *autoencoder*

Quantidade de épocas	35
Tamanho do batch	8
Taxa de aprendizagem	0.001
Quantidade total de imagens usadas para treinamento e validação	8010
Divisão do conjunto de dados	O conjunto de dados foi dividido aleatoriamente em um conjunto de treinamento com 6808 imagens (85%) e um conjunto de validação de 1202 imagens (15%)

Resultados do treinamento e validação	

Análise de similaridade

Para avaliar o grau de originalidade, realizou-se a clusterização dos dados codificados na etapa anterior, com o propósito de utilizar as correlações entre aplicativos e clusters, a fim de obter uma nota de originalidade.

A fórmula para o cálculo da nota de originalidade foi baseada na distância entre a representação codificada de uma captura de tela de aplicativo e o seu respectivo cluster, assim como a média da distância desta com os outros clusters, levando em conta todos as outras capturas de tela de aplicativo.

A distribuição das notas de similaridade de aplicativo (Figura 3) dadas pelo modelo para os aplicativos usados no treino e validação. As notas foram normalizadas para um intervalo entre 0 e 1. Analisando a distribuição no universo de referência composto de 8.010 screenshots, como esperado, existe uma curva que indica a presença de muitos aplicativos que

apresentam características semelhantes, ou seja, menos originais, e poucos aplicativos originais.

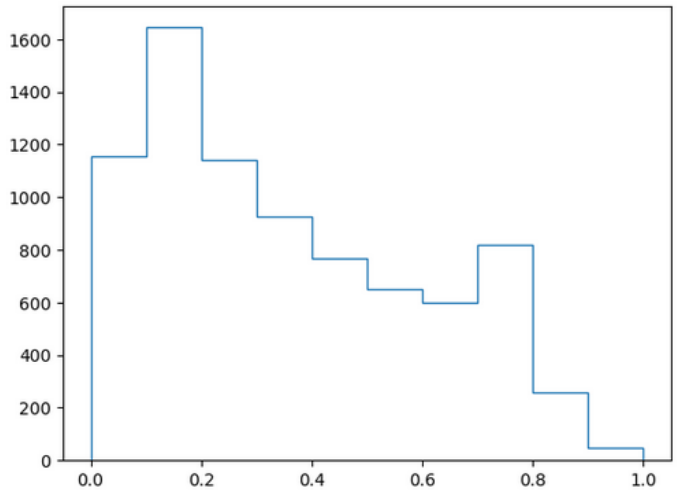


Figura 3. Distribuição de notas de originalidade no universo de referência (Fonte: próprio autor).

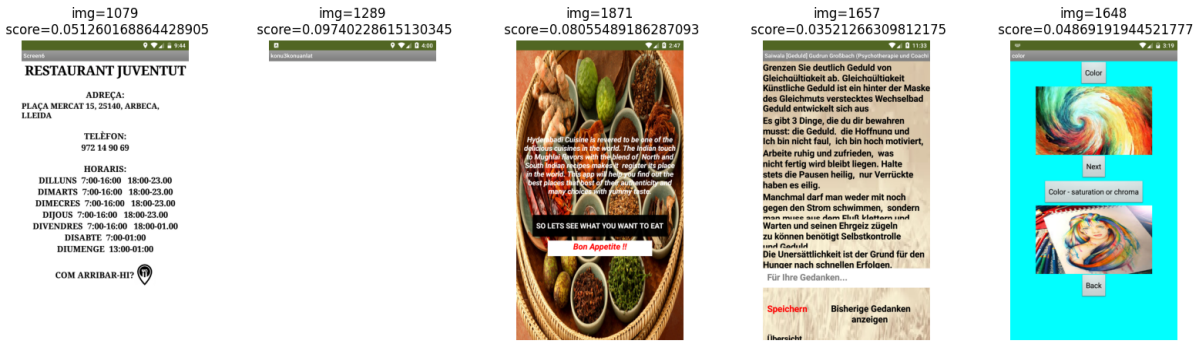


Figura 4. Exemplos de capturas de telas consideradas “não originais” pelo modelo e suas respectivas notas (Fonte: próprio autor).

Figura 5. Exemplos de capturas de telas consideradas “originais” pelo modelo e suas respectivas notas (Fonte: próprio autor).

Avaliação do desempenho

A avaliação de desempenho teve como objetivo validar o modelo, por meio da comparação entre a avaliação automatizada do modelo com a avaliação realizada por humanos. Para tal, foram selecionadas 10 capturas de tela de 10 apps diferentes. Estas 10 telas foram avaliadas por especialistas em relação a sua novidade/originalidade ranking eles de mais original (1) até menos original (10). Em paralelo é calculado automaticamente uma nota de originalidade para cada captura de tela usando o modelo desenvolvido neste trabalho.

Ao final é avaliado o desempenho do modelo com base na diferença entre o *ranking* humano e o do modelo. A fim de avaliar a correlação entre as duas variáveis foi utilizado o coeficiente de Spearman.

Para avaliar a similaridade do modelo apresentado com as notas dadas por humanos, foi utilizado o coeficiente de similaridade de Spearman [Spearman, 1904], que mede a força e direção da associação de duas variáveis. O cálculo resulta num valor entre -1 e 1, sendo que quanto mais correlacionadas as variáveis, mais próximo de 1 resulta o cálculo. Como resultado da avaliação, o coeficiente de Spearman resultou no valor -0,03 que é uma correlação negativa e próxima de 0, indicando não haver correlação entre as variáveis.

4Conclusão

Como resultado do presente trabalho foi feito um modelo que mede a similaridade de um novo app com o universo de referência, calculado ao final uma nota de originalidade. Em relação ao conjunto de testes composto por 10 apps, o modelo desenvolvido apresenta uma correlação negativa comparada com um ranking de originalidade dado por humanos, tendo em vista as razões apontadas na discussão do capítulo de avaliação do desempenho.

Considerando a natureza exploratória deste estudo sobre a avaliação de originalidade de aplicativos usando *autoencoders*, várias ameaças à validade foram identificadas. Uma dessas ameaças está relacionada à falta de uma definição mensurável e consensual do construto de originalidade. Embora exista um consenso geral sobre a importância da originalidade em produtos criativos, a literatura não oferece uma definição precisa de como medir a originalidade. Para lidar com essa ameaça, a abordagem adotada neste estudo baseou-se em definições comumente encontradas na literatura para criatividade/originalidade de aplicativos.

Outra ameaça identificada está relacionada ao tamanho da amostra. Foi utilizada uma amostra menor que a de estudos relacionados. Logo, é importante ressaltar que permanece uma questão em aberto se os resultados obtidos neste estudo seriam melhores ao considerar um universo de aplicativos consideravelmente maior.

Apesar das ameaças à validade mencionadas, este estudo proporcionou entendimento preliminar sobre o uso de *autoencoders* na avaliação de originalidade de aplicativos. Os resultados obtidos, embora limitados ao escopo e às limitações do estudo, podem fornecer uma base para pesquisas futuras e estimular discussões sobre a medição e a detecção de originalidade em layouts de aplicativos. Ainda há espaço para aprimoramentos metodológicos e expansão do estudo para uma amostra mais representativa, com o objetivo de validar e generalizar os resultados encontrados. Em suma, este trabalho teve como intuito contribuir na investigação da avaliação de originalidade de aplicativos por meio de *autoencoders*. As ameaças à validade identificadas demonstram as limitações e desafios encontrados, destacando a necessidade de pesquisas adicionais nessa área para melhorar a compreensão e a aplicabilidade dessa abordagem na prática.

References

Alves, N. C., Wangenheim, C. G., Alberto, M., Helena L. M. P. Uma Proposta de Avaliação da Originalidade do Produto no Ensino de Algoritmos e Programação na Educação Básica. Anais do Simpósio Brasileiro de Informática na Educação, Natal, Brasil, 2020.

Besemer, S., Treffinger, D. J. Analysis of Creative Products: Review and Synthesis. Journal of Creative Behavior, 15(3), 158-178 1981.

Brogden H. E., Sprecher T. B. Criteria of creativity. In Taylor C. W. (Ed.), *Creativity: Progress and potential*. New York: McGraw-Hill, 1964. Pp. 156–176.

Cavallo, D., Gomes, A. S., Singer, H., Ibert, I. B. Inovação e Criatividade na Educação Básica: Dos conceitos ao ecossistema. Revista Brasileira de Informática na Educação, 24(2), 2016.

Ferreira, M. V. R., Wangenheim, C. G. Alves, N. C.. Desenvolvimento de um Modelo de Avaliação da Originalidade de Aplicativos Móveis Usando Técnicas de *Machine Learning*. Relatório técnico, INCoD-GQS.064.2021P. 2021, INCoD/INE/UFSC, Florianópolis, Brasil, 2021.

Ferreira, M. N. F., Pinheiro, F. da C., Wangenheim, C. G., C., Filho, R. M., Hauck, J. C. R. Ensinando design de interface de usuário de aplicativos móveis no ensino fundamental. Revista Brasileira de Informática na Educação, 28(0), 2020.

Kingma P. D., Ba. J.. Adam: A Method for Stochastic Optimization. 3rd ICLR, San Diego, 2015.

Patton, E. W., Tissenbaum M., Harunani F.. Computational Thinking Education. MIT, Cambridge, 2019.

Souza, A. S., Alves, N. C., Wangenheim, C. G., Kreuch, L. Análise Automatizada da Originalidade de Design de Interfaces de Usuário no Contexto Educacional: Um Mapeamento da Literatura. Anais do Simpósio Brasileiro de Informática na Educação, Porto Alegre, Brasil, 2021.