UNIVERSIDADE FEDERAL DE SANTA CATARINA - CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Desenvolvimento de um Modelo de Geração Automática de *Wireframes* no App Inventor a partir de *Sketches* usando *Deep Learning*

Daniel de Souza Baulé

Florianópolis

2020

Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Desenvolvimento de um Modelo de Geração automática de *Wireframes* no App Inventor a partir de *Sketches* usando *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.

Autor: Daniel de Souza Baulé

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

Florianópolis

2020

Daniel de Souza Baulé

Desenvolvimento de um Modelo de Geração automática de Wireframes no App Inventor a partir de Sketches Usando 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.

Florianópolis, 01 de novembro de 2020

Prof.ª Dr.ª rer. nat. Christiane Gresse von Wangenheim, PMP
Orientadora
Universidade Federal de Santa Catarina

Prof. Dr. rer.nat. Aldo von Wangenheim
Coorientador
Universidade Federal de Santa Catarina

Prof. Dr. Mauro Roisenberg
Avaliador
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Gostaria de agradecer a todos os participantes na criação do conjunto de dados pelos *sketches* desenhados, e a todos os voluntários que responderam o estudo de usuário, por sua ajuda na avaliação da ferramenta, bem como o *feedback* e sugestões fornecidos.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACM - Association for Computing Machinery

AP - Average Precision

CNN – Convolutional Neural Network

DL – Deep Learning

DNN – Deep Neural Network

GUI – Graphical User Interface

IA – Inteligência Artificial

IEEE - Institute of Electrical and Electronics Engineers

IoU – Intersection Over Union

NI – Não Informada

NN – Neural Network

SUS – System Usability Scale

UI – User Interface

RESUMO

Nos últimos anos os aplicativos para dispositivos móveis se tornaram parte fundamental do cotidiano da população, sendo utilizados em diversos aspectos da vida profissional e pessoal. Esse crescimento de sua importância causou um aumento no número de aplicativos do mercado, tornando-o muito competitivo. Isso tem gerado ênfase cada vez maior em fatores decisivos, como a interação do usuário com o aplicativo. Assim, a prototipação de interface gráfica dos aplicativos tem ganhado cada vez mais importância no processo de desenvolvimento. Durante o processo de prototipação da interface de usuários são criados *sketches*, que são utilizados para a geração de *wireframes*, representações de baixa fidelidade da interface. A conversão destes *sketches* para *wireframes* de forma manual é um processo custoso e demorado. Com essa motivação estão sendo criados ferramentas que automatizam essa transformação. Porém, atualmente ainda não existem ferramentas que geram *wireframes* para App Inventor, um ambiente de programação visual e intuitivo que permite qualquer pessoa criar um aplicativo Android. Neste contexto, o objetivo deste trabalho é o desenvolvimento de um sistema, utilizando técnicas de *deep learning,* para geração automática de *wireframes* no App Inventor a partir de *sketches*, visando facilitar o processo de prototipação e desenvolvimento de interfaces gráficas.

Palavras-chave: App Inventor, *Sketch*, *Wireframe*, *Design* de Interface de Usuário, *Deep Learning*.

1. INTRODUÇÃO

1.1. Contextualização

Nos últimos anos os aplicativos para dispositivos móveis têm se tornado cada vez mais presentes na vida de todos (FU et al., 2013). Por todo mundo, aplicativos são utilizados para as mais diferentes funções, desde entretenimento até monitoramento de saúde (SCHLATTER et al., 2013). Um importante aspecto da utilização de aplicativos é a sua interação com usuário (NEIL, 2014). A usabilidade de um aplicativo é um importante fator determinante de seu sucesso, com sua interface ocupando um papel fundamental nesta interação (NEIL, 2014). Ter uma interface intuitiva e atraente deixou de ser simplesmente um adicional, passando a ser requisito para um aplicativo obter um número considerável de usuários (NEIL, 2014).

Com o aumento da importância da interface gráfica, o desenvolvimento destas ganhou maior destaque no projeto de aplicativos (NEIL, 2014). Porém, o desenvolvimento de interfaces gráficas de qualidade não é um processo simples, exigindo um esforço e recursos consideráveis (ROBINSON, 2018).

Tipicamente, o desenvolvimento de interfaces gráficas envolve um processo de prototipação, iniciado com um processo criativo, fazendo *sketches*, representações simples, geralmente feitas com papel e caneta (SCHLATTER et al., 2013). *Sketches* são um meio visual efetivo para transmitir ideias de maneira simples e rápida, sendo utilizados para expandir ideias, visualizar conceitos abstratos e comparar diferentes alternativas (HUANG et al., 2019). Os *sketches* são um componente importante da fase de desenvolvimento da interface, pois facilitam a realização e teste de mudanças na interface bem como a visualização de seu impacto (HUANG et al., 2019).

A partir dos *sketches* são criados *wireframes* de interface, artefatos de design de baixa fidelidade que definem a estrutura base da interface sem detalhes do design

visual como cores etc. (ROBINSON, 2018). A criação dos *wireframes* é tipicamente realizada manualmente utilizando ferramentas de *design* (ROBINSON, 2018). Após a criação de um *wireframe*, este é revisado e o *design* visual é integrado, até se tornar um protótipo de alta fidelidade. Ao ser finalizado o *design*, este é implementado, resultando no produto (ROBINSON, 2018).

Por se tratar de um processo de prototipação iterativo, estas etapas podem se repetir diversas vezes durante o *design* de uma interface, sendo necessário o retrabalho sempre que sejam realizadas mudanças (ROBINSON, 2018). Assim, o desenvolvimento de design de interface é uma etapa que consome um esforço e tempo considerável (ROBINSON, 2018). Nesse contexto, especificamente a geração de *wireframes* a partir de *sketches* é um processo com alto potencial de automatização, resultando numa redução de esforço e tempo desta etapa no processo de prototipação de interface de usuário (ROBINSON, 2018).

Atualmente estão sendo criadas algumas soluções para a geração automática de *wireframes* para *websites* e aplicativos móveis, possibilitando a geração automática de *wireframes* em HTML, Sketch, React, Android Studio entre outros (ROBINSON, 2018; BELTRAMELLI, 2017). Utilizando técnicas variadas, como visão computacional clássica (HUANG et al., 2019), redes neurais convolucionais (YUN et al., 2018) ou alguma combinação destas técnicas (ROBINSON, 2019).

Técnicas de *deep learning* se tornaram conhecidas nos últimos anos por sua excelente performance em reconhecimento de imagem e classificação de objetos (PATHMIND, 2019a). Em particular redes neurais convolucionais são utilizadas para reconhecimento e classificação de caracteres manuscritos (PATHMIND, 2019b), indicando seu grande potencial para detecção e classificação de elementos de interface em *sketches* (PATHMIND, 2019b).

Apesar da existência de ferramentas similares, nenhuma ferramenta disponível permite a geração de *wireframes* para App Inventor, um ambiente de programação visual e intuitivo que permite qualquer pessoa criar um aplicativo para Android (MIT, 2019). A facilidade de uso do App Inventor o torna uma poderosa ferramenta tanto para o ensino de programação quanto para a realização de projetos de aplicativos por parte de *end-users*, profissionais de diversas áreas, sem necessidade de familiaridade com linguagens de programação e ferramentas mais avançadas (MIT, 2019). Tal facilidade de uso aumenta a acessibilidade da criação de aplicativos, permitindo que diferentes ideias por todo o mundo sejam finalmente colocadas em prática (WOLBER et al., 2015). Assim, a criação de uma ferramenta para geração automática de *wireframes* no App Inventor a partir de *sketches* permitirá com mais facilidade a adoção de um processo sistemático de prototipação do design de interface de usuário de apps sendo criado com App Inventor.

1.2. Objetivos

Objetivo geral

O objetivo geral deste trabalho é desenvolver e testar um modelo de geração automática de *wireframes* de apps Android no App Inventor a partir de *sketches*. São adotadas técnicas de *Deep Learning* para criar, treinar e testar uma rede neural capaz de realizar a conversão dos *sketches* em modelos semânticos e em seguida a geração do *wireframe* no App Inventor a partir do modelo semântico obtido.

Objetivos Específicos

O1. Analisar a fundamentação teórica sobre aprendizagem de elementos de design de interface de usuário e *deep learning*.

O2. Analisar o estado da arte em relação a automação da conversão de sketches em *wireframes* de apps Android;

O3. Desenvolver e testar um modelo de geração de modelo semântico a partir de sketches;

O4. Desenvolver e testar a geração de código do App Inventor no nível *wireframe* a partir do modelo semântico.

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 proposto tem como foco a geração automática de *wireframes*, não abordando outros detalhes da interface. O trabalho foca na análise de aplicativos Android não abordando interfaces de apps de outras plataformas. A automatização enfocará somente no desenvolvimento de projetos com a ferramenta App Inventor.

1.3. Metodologia de pesquisa

A metodologia de pesquisa aplicada 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 descrita por Pizzani et al. (2012). Nesta etapa são realizadas as seguintes atividades:

A1.1 – Análise teórica sobre design de interfaces de usuário;

A1.2 – Síntese de conceitos de design de interface do App Inventor;

A1.3 – Síntese de conceitos básicos sobre *deep learning*.

Etapa 2 – Estado da arte

Nesta etapa é realizado um mapeamento sistemático da literatura seguindo o processo proposto por Petersen et al. (2008) para identificar e analisar modelos de geração automatizada de *wireframes* de interfaces atualmente publicados. 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 do modelo semântico

Nesta etapa é desenvolvido um modelo para geração automática de *wireframes* a partir de *sketches*, seguindo iterativamente um processo de desenvolvimento de redes neurais/*deep learning* (KIERSKI, 2017; SHUAI, 2017; POLYZOTIS et al., 2017). Esta etapa é dividida nas seguintes atividades:

A3.1 – Análise de requisitos;

A3.2 – Coleta de dados;

A3.3 – Preparação de conjunto de dados;

A3.4 – Seleção de um modelo de rede neural;

A3.5 – Treinamento da rede neural;

A3.6 – Avaliação da rede neural;

A3.7 – Afinação de parâmetros;

A3.8 – Predição/Inferência.

Etapa 4 – Desenvolvimento do código de App inventor

Nesta etapa é desenvolvido o modulo para transformar o modelo semântico em código do App Inventor, seguindo um processo de engenharia de software proposto por Pressman (2016). Esta etapa é dividida nas seguintes atividades:

A4.1 – Análise de requisitos;

A4.2 – Modelagem da arquitetura do sistema;

A4.3 – Modelagem detalhada e implementação;

A4.4 – Testes do sistema.

1.4. Estrutura do documento

No capítulo 2 é apresentada a fundamentação teórica dos conceitos necessários que sustentam a proposta deste trabalho. No capítulo 3 é apresentado o estado da arte, a situação atual em que se encontram os trabalhos e propostas existentes na área de geração automática de *wireframes* no App Inventor a partir de *sketches* usando *deep learning*. O capítulo 4 apresenta os resultados criados no desenvolvimento do modelo “Sketch2aia” de geração automática de *wireframes* para App Inventor a partir de *sketches*. O capítulo 5 apresenta a avaliação do modelo por meio de um estudo de usuário. O capítulo 6 apresenta os resultados e comparações

das análises de desempenho do modelo desenvolvido. Para finalizar, o capítulo 7 apresenta a conclusão do presente trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo são apresentados os conceitos relevantes ao trabalho desenvolvido. Iniciando por uma análise teórica da ferramenta App Inventor, seguido por uma análise teórica do processo de *design* de interfaces de aplicativos móveis, e por fim uma análise teórica sobre inteligência artificial e redes neurais, com foco em *deep learning.*

2.1. App Inventor

Com o crescimento da utilização de computadores e celulares pelo mundo, vem o aumento do interesse pela criação de aplicativos não apenas por profissionais da área, mas também por indivíduos sem educação formal ou conhecimentos em áreas de computação e programação (WOLBER et al., 2015). Visando facilitar a aprendizagem de programação, diversas linguagens de programação visual foram criadas nos últimos anos, como Scratch (2018), Snap! (2018) ou Blockly (2019). Estas linguagens tornam a programação mais acessível, pois a aplicação é desenvolvida por meio da ordenação de blocos visuais pelo usuário, sem necessidade de compreender e memorizar sintaxes de linguagens textuais (CSTA, 2017).

O App Inventor (MIT, 2019) é uma ferramenta de código-aberto, desenvolvida na Google com a intenção de permitir que “as pessoas se tornem criadores, e não apenas consumidores” (PATTON et al., 2019), sendo mantido atualmente pelo Instituto de Tecnologia de Massachusetts (MIT). O App Inventor auxilia no desenvolvimento de aplicativos (ou apps) para dispositivos Android com suporte para *design* de interface e para a programação funcional do aplicativo. Este ambiente de programação permite proporcionar às pessoas com pouco ou nenhum conhecimento de programação a experiência de desenvolver aplicativos para dispositivos Android

de forma intuitiva. Em 2019, o App Inventor tem mais de 500 mil usuários ativos por mês de 195 países (MIT, 2019).

O desenvolvimento de um aplicativo no App Inventor é dividido em duas áreas: Designer e Blocos. A área Designer é utilizada para a configuração dos componentes de Interface e de conectividade de recursos do celular (Figura 1), como botões, imagens, sons, sensores, temporizadores, mapas e conexões *bluetooth* entre outros.

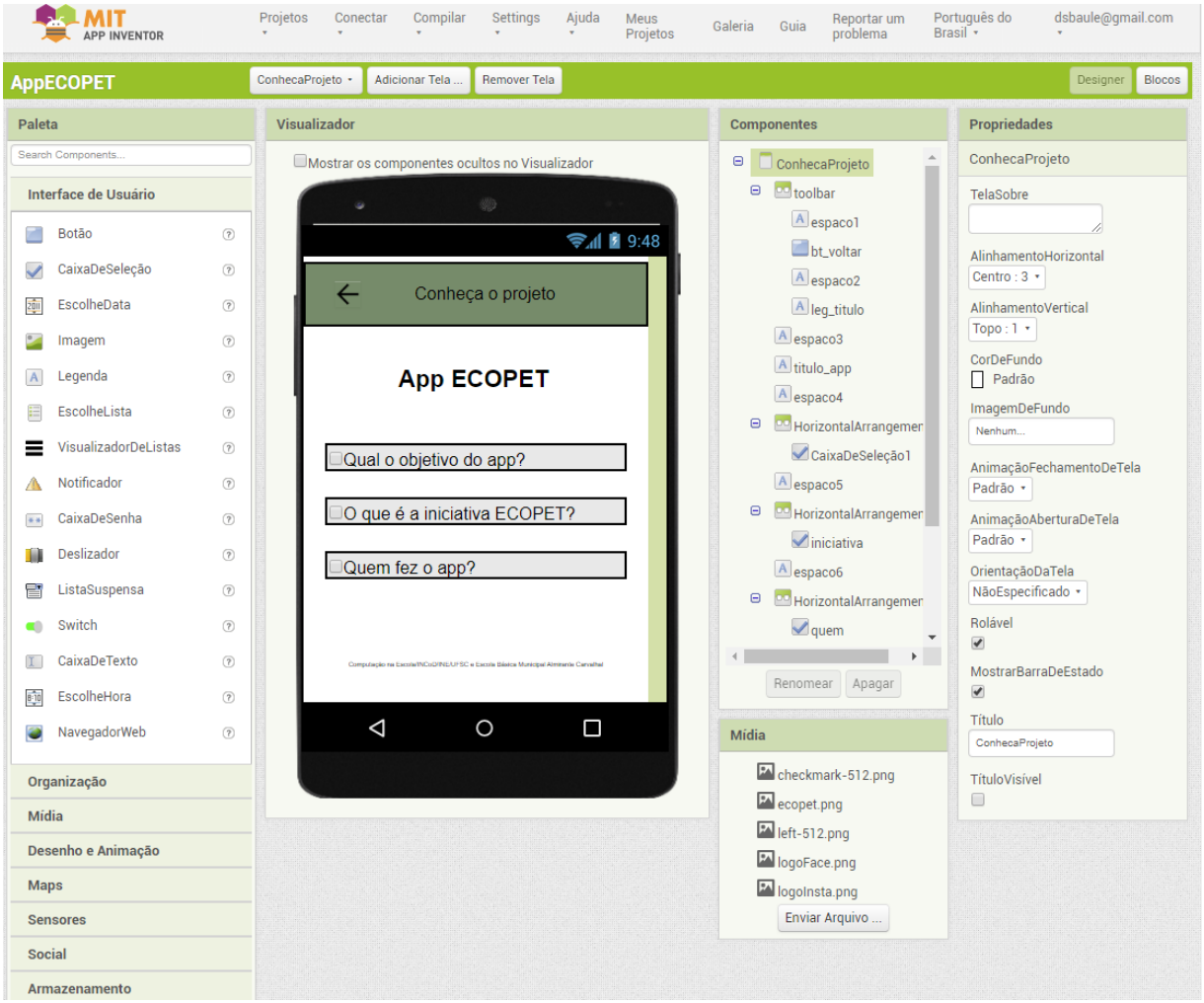


Figura 1: Tela "Designer" do App Inventor (MIT, 2019)

A área Blocos (Figura 2) é utilizada para a programação da lógica do aplicativo, por meio de funções, condicionais, laços etc. Para a interação da lógica com os componentes da interface, o App Inventor apresenta blocos que podem ser usados para cada componente adicionados nas telas.

Figura 2: Tela "Blocos" do App Inventor (MIT, 2019)

O desenvolvimento da parte funcional do aplicativo é feito com programação visual baseada em blocos. Para isso, o App Inventor utiliza a biblioteca de código aberto *Blockly* (2019) para construir o vocabulário de blocos de programação que podem ser utilizados (PASTERNAK; FENICHEL; MARSHALL, 2017).

O App Inventor disponibiliza diversos componentes que podem ser utilizados para desenvolvimento do aplicativo, permitindo melhor explorar as funcionalidades do celular (Tabela 1).

Tabela 1: Componentes do Designer do App Inventor

Grupo	Componentes	Descrição
Interface de Usuário	Botão; CaixaDeSeleção; EscolheData; Imagem; Legenda; EscolheLista; VisualizadorDeListas; Notificador; CaixaDeSenha; Deslizador; ListaSuspensa; Switch; CaixaDeTexto; EscolheHora; NavegadorWeb.	Componentes principais para interação do usuário.
Organização	OrganizaçãoHorizontal; HorizontalScrollArrangement; OrganizaçãoEmTabela; OrganizaçãoVertical; VerticalScrollArrangement.	Auxiliam na organização dos demais componentes.
Mídia	CâmeraDeVídeo; Câmera; EscolheImagem; Tocador; Som; Gravador; ReconhecedorDeVoz; TextoParaFalar; ReprodutorDeVídeo; TradutorYandex.	Componentes para gravação e reprodução de mídia (Som, Imagem ou Vídeo).
Desenho e Animação	Bola; Pintura; Sprite; Imagem.	Componentes que permitem que o usuário desenhe e visualiza animações
Maps	Circle; FeatureCollection; LineString; Map; Marker; Polygon; Rectangle.	Um componente mapa, e demais componentes para sua personalização.
Sensores	SensorAcelerômetro; CódigoDeBarras; Temporizador; GyroscopeSensor; SensorDeLocalização; NearField; SensorDeOrientação; Pedometer; SensorDeProximidade.	Componentes para obtenção de informações dos sensores do dispositivo (celular).
Social	EscolheContato; EscolheEmail; Ligação; EscolheNúmeroDeTelefone; Compartilhamento; MensagensSMS; Twitter.	Componentes para comunicação com aplicativos sociais.
Armazenamento	CloudDB; Arquivo; ControleDeFusiontables; TinyDB; TinyWebDB.	Criação de arquivos e bancos de dados.
Conectividade	IniciadorDeAtividades; ClienteBluetooth; ServidorBluetooth; Web.	Conexão da aplicação com outros dispositivos.
LEGO® MINDSTORMS®	NxtSensorDeCor; NxtSensorDeLuz; NxtSensorDeSom; NxtSensorDeToque; NxtSensorUltrasônico; NxtComandosDiretos; Ev3Motors; Ev3ColorSensor; Ev3GyroSensor; Ev3TouchSensor; Ev3UltrasonicSensor; Ev3Sound; Ev3UI; Ev3Commands.	Componentes para interface com a plataforma LEGO® MINDSTORMS®.
Experimental	FirebaseDB.	Componentes em fase experimental.
Extension		Vazio por padrão, utilizado para importação de componentes por meio de extensões.

O App Inventor disponibiliza ao usuário diferentes componentes de interface de usuário que podem ser livremente posicionados. Cada um dos componentes disponibilizado possui funções e características específicas (Tabela 2).

Tabela 2 - Componentes de design de interface de usuário disponíveis no App Inventor (2019)

Componente	**Descrição**
Botão	Botão com a habilidade de detectar cliques.
Caixa de Seleção	Caixa de seleção que pode disparar um evento quando o usuário clica nela.
Escolhe Data	Um botão que, quando clicado, inicia um diálogo que permite ao usuário selecionar uma data.
Imagem	Componente para apresentação de imagens.
Legenda	Uma Legenda mostra um trecho de texto, que é especificado por meio da propriedade Texto.
Escolhe Lista	Um botão que, quando clicado, mostra uma lista de textos para o usuário escolher.
Visualizador De Listas	Este é um componente visível que permite colocar uma lista de elementos de texto para apresentação na sua Tela.
Notificador	O componente Notificador mostra diálogos de alerta, mensagens, e alertas temporários, e cria entradas de log Android.
Caixa De Senha	Uma caixa para digitar senhas. O mesmo que o componente CaixaDeTexto comum, exceto por não exibir os caracteres digitados pelo usuário.
Deslizador	Um Deslizador é uma barra de progresso que adiciona um indicador arrastável. Você pode tocar no indicador e arrastar para a esquerda ou para a direita para ajustar sua posição.
Lista Suspensa	Um componente para seleção que exibe um *pop-up* com uma lista de elementos.
Switch	Interruptor que gera um evento ao ser clicado pelo usuário.
Caixa De Texto	Uma caixa para o usuário inserir texto. O valor inicial ou texto inserido pelo usuário está na propriedade Texto.
Escolhe Hora	Um botão que, quando clicado, inicia um diálogo que permite ao usuário selecionar um horário.
Navegador Web	Componente para a visualizar páginas da web.

É possível também alterar as configurações de cada componente de UI. Por exemplo, ao adicionar um botão no aplicativo, é possível:

- Escolher a cor de fundo;
- Formatar o texto apresentado (como itálico, negrito, tamanho da fonte, tipo da fonte, cor, alinhamento);
- Tamanho do componente (altura e largura);
- Habilitação se é inicialmente visível ou não, dentre outras.

O App Inventor permite também a exportação/importação de projetos por meio de arquivos compactados no formato aia. Na versão nb179 do App Inventor, este arquivo inclui todas as informações do app, desde imagens e sons utilizados até os

códigos produzidos pelos blocos da lógica, componentes visuais presentes no app e propriedades do projeto do App Inventor (Figura 3).

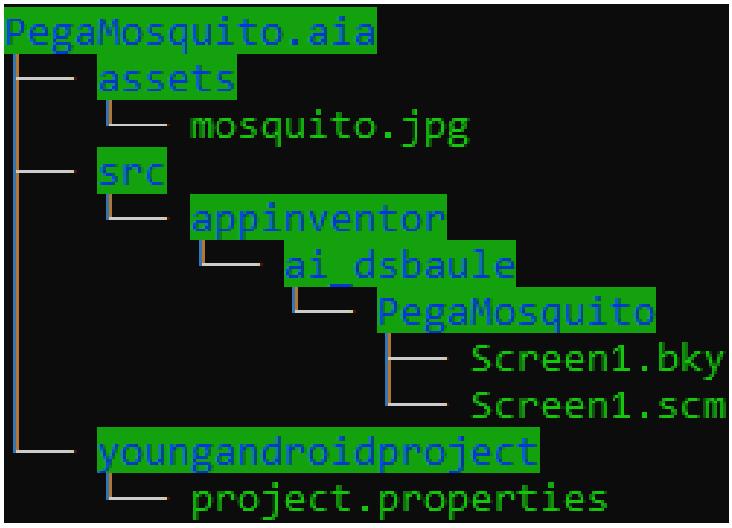


Figura 3: Estrutura de um arquivo .aia exemplo

Em um arquivo .aia, a pasta *"assets"* contém os elementos de mídia necessários (imagens, sons etc.) enquanto os códigos referentes a parte visual do app ficam registrados em arquivos no formato .scm que por sua vez encapsulam uma estrutura json com todos os componentes utilizados. Os códigos produzidos pela parte lógica do app ficam registrados em arquivos no formato .bky que por sua vez encapsulam estruturas xml com toda a lógica implementada. Por fim, um arquivo .properties contém propriedades do projeto no App Inventor adicionais.

2.2. *Design* de interfaces de aplicativos móveis

2.2.1. Interfaces Gráficas

No processo de desenvolvimento de um aplicativo, uma das etapas com maior impacto na experiência do usuário é o *design* de sua interface (FU et al., 2013). É com a interface gráfica que o usuário interage diretamente, logo, é muito importante que no processo sejam consideradas as características do usuário e como este irá utilizar

o software. O objetivo deste processo é o *design* de uma interface interativa, fácil de usar, efetiva e agradável na visão do usuário (ROGERS; SHARP; PREECE,2013).

O processo de *design* de uma interface não é algo trivial, sendo um processo complexo e iterativo, testando diversas ideias, criando diferentes versões de interface e melhorando-as até a criação da interface final (ROBINSON, 2018).

Este processo geralmente inicia com a realização de *sketches* manuais ou *mockups, sketches* feitos com auxílio de ferramentas computacionais (MORAN et al., 2018). São realizados diversos *sketches* como uma forma de transmitir ideia e testar conceitos (HUANG et al., 2019). Uma vez escolhido um *design*, o *sketch* é passado para um programador que implementa o *wireframe* correspondente, com os detalhes visuais sendo adicionados posteriormente, construindo-se um protótipo de alta fidelidade. Se o protótipo for aceito, se torna um produto, mas a qualquer momento no processo podem ocorrer alterações no *design*, voltando a fase de *sketches,* como mostra a Figura 4.

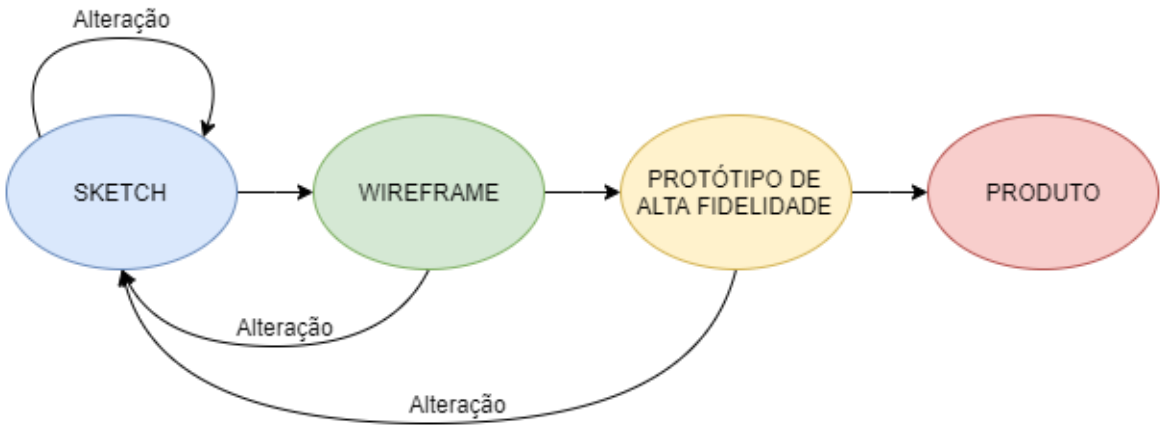


Figura 4 - Processo iterativo de design de interface

A Figura 5 mostra um exemplo deste processo, com os artefatos criados no *design* do aplicativo Bikanto (MISSFELDT FILHO et al., 2017).

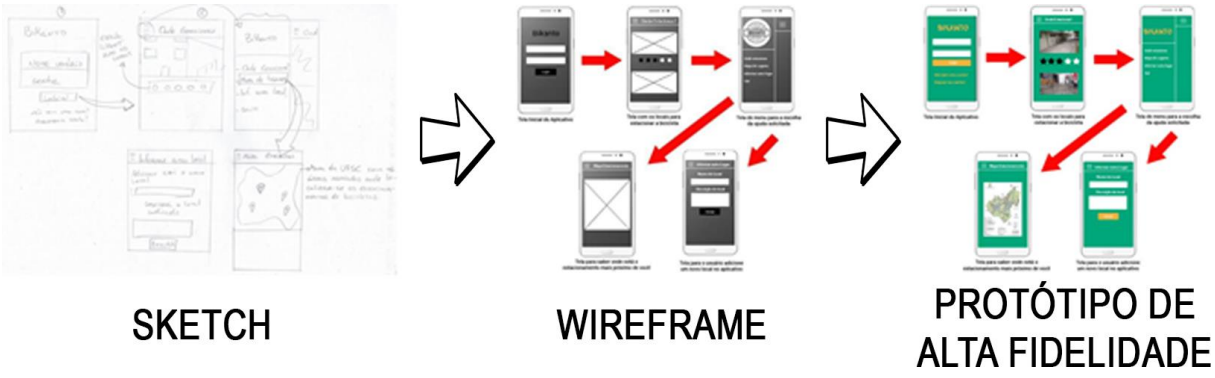


Figura 5 - Processo de *Design* de Interface do Aplicativo Bikanto (MISSFELDT FILHO et al., 2017)

2.2.2. Criação de *Sketches*

Tipicamente o processo de criação de uma interface de usuário iniciam com rascunho de telas, conhecidos como *sketches* (LANDAY; MYERS, 1994), representações simples, geralmente feitas com papel e caneta (Figura 6). *Sketches* são um meio visual efetivo para transmitir ideias de maneira simples e rápida, sendo utilizados para expandir ideias, visualizar conceitos abstratos e comparar diferentes alternativas (HUANG et al., 2019).

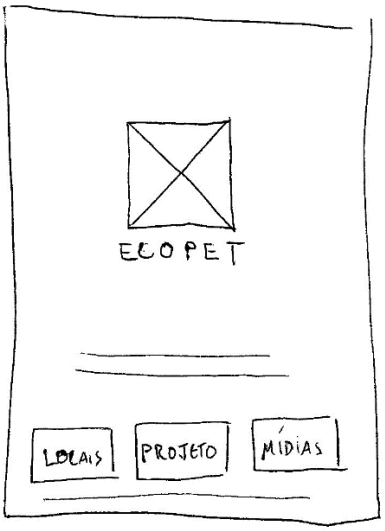
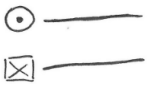
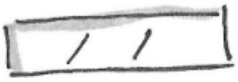
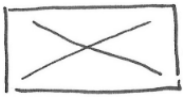
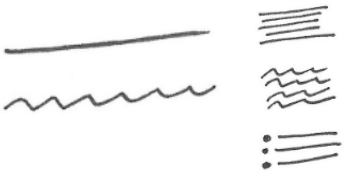
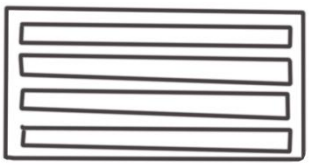
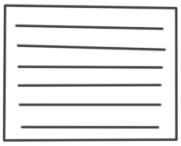
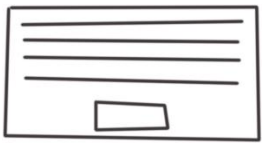


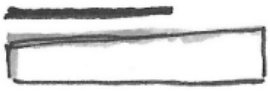
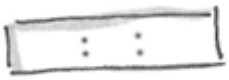
Figura 6 - *Sketch* de uma tela do aplicativo ECOPET (GQS, 2019)

Sketches são utilizados para trabalhar sobre o layout dos componentes, sem se preocupar com os detalhes, permitindo a comparação de diferentes ideias de interface de maneira rápida e intuitiva (LANDAY; MYERS, 1994).

Para a criação de *Sketches*, é necessário definir representações simplificadas dos diferentes elementos de interface, de modo a facilitar o desenho destes. A Tabela 3 mostra exemplos da aparência de diferentes componentes em *Sketches*, parcialmente com base em Kas (2019).

Tabela 3 - Aparência em *Sketches* dos diferentes Componentes de Interface do App Inventor

Componente	**Aparência em *Sketches***
Botão	
Caixa de Seleção	
Escolhe Data	
Imagem	
Legenda	
Escolhe Lista	
Visualizador De Listas	
Notificador	
Caixa De Senha	
Deslizador	
Lista Suspensa	

Switch	
Caixa De Texto	
Escolhe Hora	
Navegador Web	

Devido a sua informalidade, as representações em *Sketches* podem apresentar pequenas diferenças dependendo do indivíduo que a desenhou, mas a definição de algum padrão compreensivo se torna necessária para a correta transmissão de ideias, ainda que este aceite algumas variações. A Figura 7 mostra outro exemplo de *sketches*, com os *sketches* utilizados para *design* do aplicativo ClicDenúncia (GRESSE VON WANGENHEIM et al., 2017).

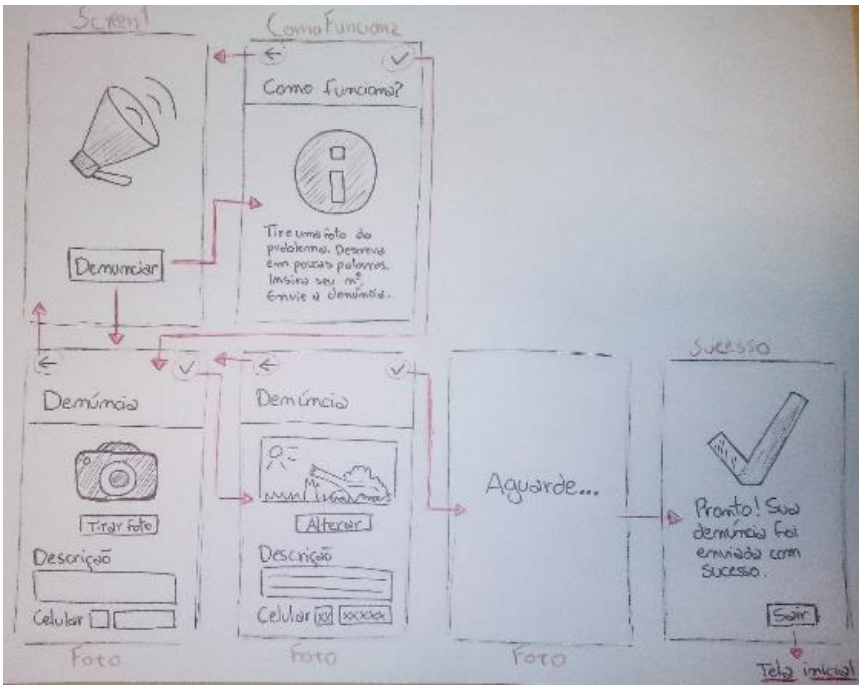


Figura 7 - *Sketches* de Interface do Aplicativo ClicDenúncia (GRESSE VON WANGENHEIM et al., 2017)

2.2.3. Geração de *Wireframes*

Uma vez que um *sketch* é aprovado, cabe a um programador a implementação de um *wireframe* de interface em uma ferramenta computacional, para que as funcionalidades do aplicativo possam ser implementadas (SCHLATTER et al., 2013). *Wireframes* são representações hierárquicas da interface, apenas com informações básicas como tipos de elementos e suas posições, sem detalhes de *design* visual (SCHLATTER et al., 2013). A Figura 8 mostra um exemplo de *wireframe*, gerado durante o *design* do aplicativo Bikanto (MISSFELDT FILHO et al., 2017).

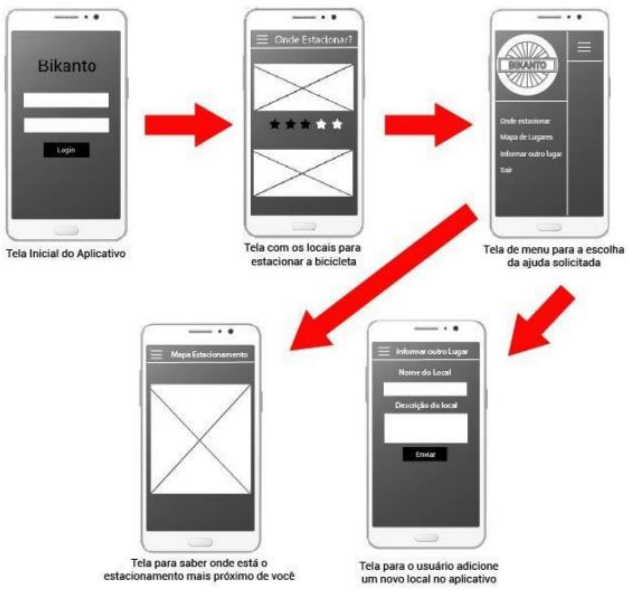
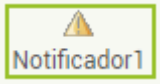
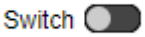
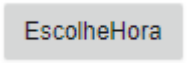


Figura 8 - *Wireframes* de Interface do Aplicativo Bikanto (MISSFELDT FILHO et al., 2017).

A aparência dos elementos de interface em *wireframes* varia de acordo com o programa utilizado. A Tabela 4 mostra a aparência dos componentes de interface em *wireframes* de App Inventor.

Tabela 4 - Aparência em *Wireframes* dos diferentes Componentes de Interface do App Inventor

Componente	**Aparência em *Sketches***
Botão	Botão
Caixa de Seleção	CaixaDeSeleção

Escolhe Data	
Imagem	
Legenda	Legenda
Escolhe Lista	
Visualizador De Listas	
Notificador	 (Fora da Tela)
Caixa De Senha	
Deslizador	
Lista Suspensa	
Switch	
Caixa De Texto	
Escolhe Hora	
Navegador Web	

É possível observar que o App Inventor não utiliza aparências únicas para alguns componentes, com componentes como EscolheData, EscolheLista e EscolheHora, por exemplo, sendo visualmente iguais a um botão. Isso não apresenta um problema, uma vez que na representação da hierarquia da interface o tipo de

componente é informado explicitamente, apenas dificulta a compreensão da interface com base exclusivamente em uma inspeção visual do *wireframe*.

Quando um *wireframe* é aceito e teve suas funcionalidades testadas com sucesso, inicia a próxima etapa do processo. Elementos de *design* visual são adicionados, resultando em um protótipo de alta fidelidade (ROBINSON, 2018). A Figura 9 mostra um protótipo de alta fidelidade do aplicativo Bikanto.

Figura 9 - Protótipo de alta fidelidade do aplicativo Bikanto (MISSFELDT FILHO et al., 2017).

O protótipo de alta fidelidade é um artefato próximo ao produto. Se nenhum problema for detectado, são realizados pequenos aprimoramentos de *design*, e é finalizado o aplicativo. A Figura 10 mostra a interface final do aplicativo Bikanto.

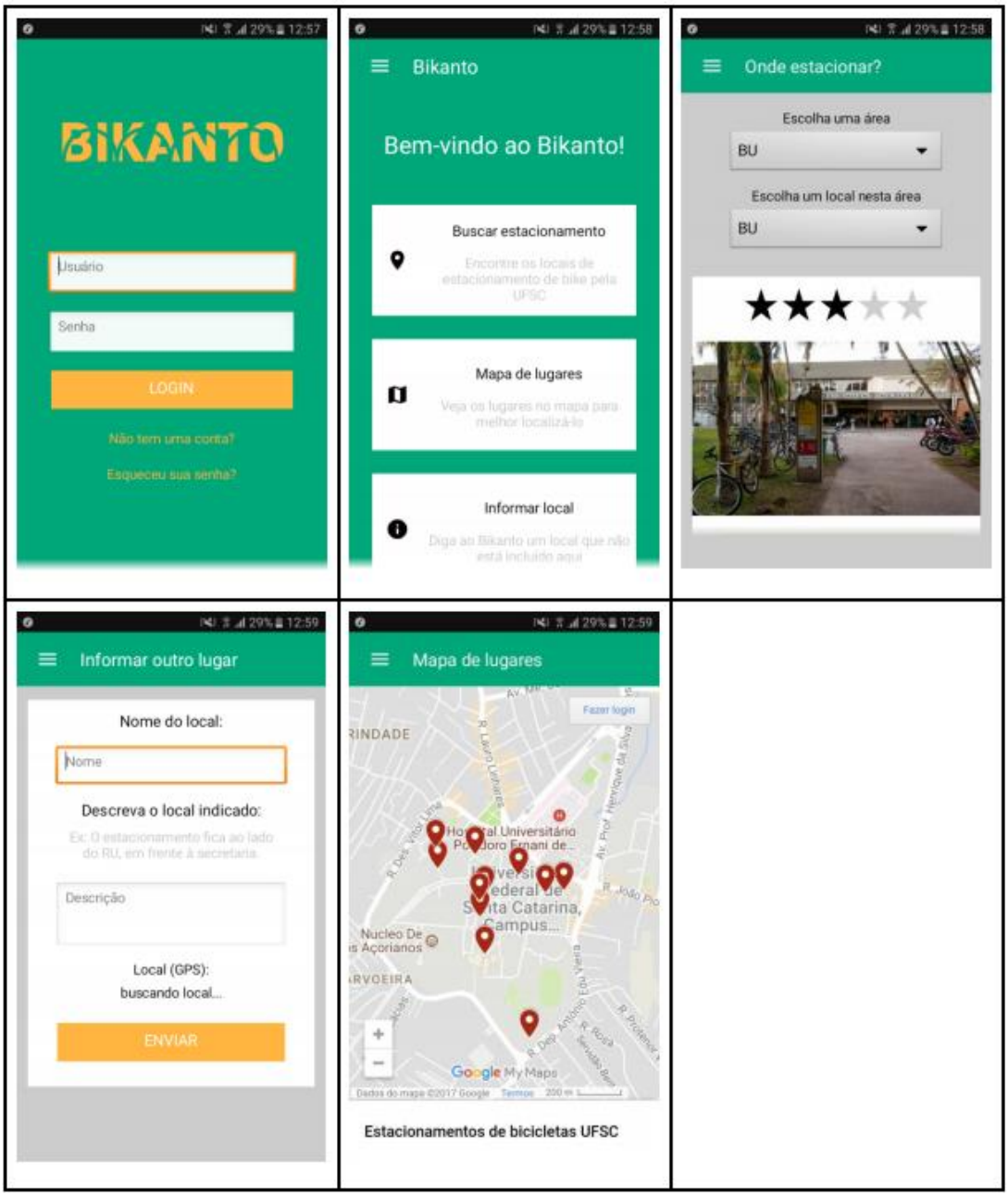


Figura 10 - Resultado das telas do aplicativo Bikanto (MISSFELDT FILHO et al., 2017).

2.3. Deep Learning

John McCarthy, reconhecido como um dos pais da Inteligência Artificial (IA), definiu IA como "*a ciência e engenharia de criar máquinas inteligentes que tem habilidade para alcançar objetivos da mesma forma que humanos*" em 1995. Ou seja, Inteligência Artificial é inteligência humana demonstrada por máquinas (INDIA, 2018).

Arthur Samuel definiu *Machine Learning (ML)* em 1959 como o grande subcampo da área de IA que lida com o campo de estudos que dá a computadores a habilidade de aprender sem ser explicitamente programados (INDIA, 2018) (Figura 11).

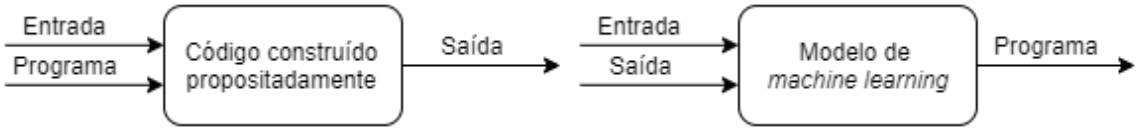


Figura 11 - Código construído propositadamente vs modelo *machine learning* (adaptado a partir de INDIA, 2018).

Machine Learning permite que se aprenda constantemente com os dados, de modo a possivelmente prever mudanças futuras. Este conjunto de algoritmos e modelos tem sido usado em diversos mercados como forma de melhorar processos e ganhar aprofundamento em padrões e anomalias em dados (INDIA, 2018).

Deep Learning permite que modelos computacionais com múltiplas camadas de processamento aprendam representações de dados com múltiplos níveis de abstração (LECUN et al., 2015). Estes métodos melhoraram drasticamente o estado da arte em reconhecimento de voz, reconhecimento visual de objetos, detecção de objetos e muitos outros domínios como pesquisa em drogas e genética (LECUN et al., 2015).

Deep Learning descobre estruturas delicadas em grandes conjuntos de dados, trazendo revoluções no processamento de imagens, vídeo, fala e áudio (LECUN et al., 2015). Isso ocorre pois aprende a representar o mundo como uma hierarquia aninhada de conceitos, com cada conceito definido em relação a outro mais simples, e representações abstratas computadas em termos de outras menos abstratas (GOODFELLOW et al., 2016).

A Figura 12 ilustra o relacionamento das diferentes disciplinas de IA, mostrando que *deep learning* é um tipo de *feature learning*, que por sua vez é um tipo de *machine learning*, que é utilizado por várias, mas não todas, as abordagens a IA. Já a Figura 13 provém um fluxograma de alto nível sobre como cada uma funciona, evidenciando as similaridades entre seus funcionamentos e como as capacidades de aprendizado a partir de entradas mais simples parecem se tornar maiores na direção de *deep learning*.

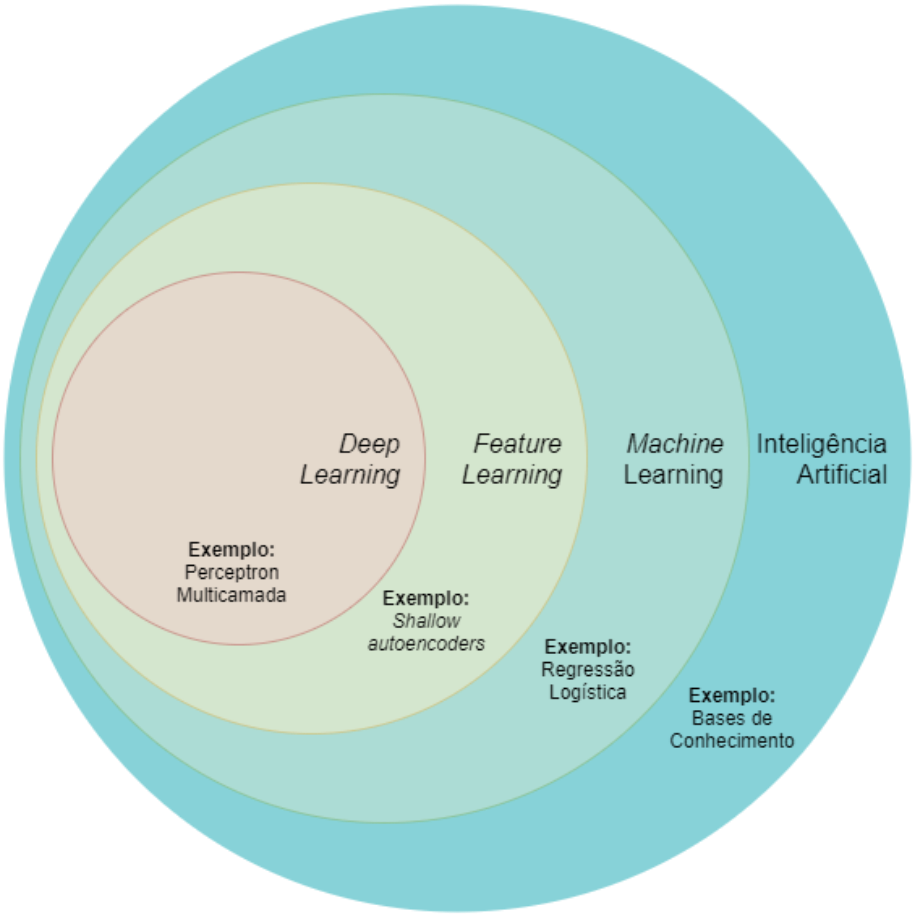


Figura 12 - Relação entre *Deep Learning* e a área de inteligência artificial (adaptado a partir de GOODFELLOW et al., 2016).

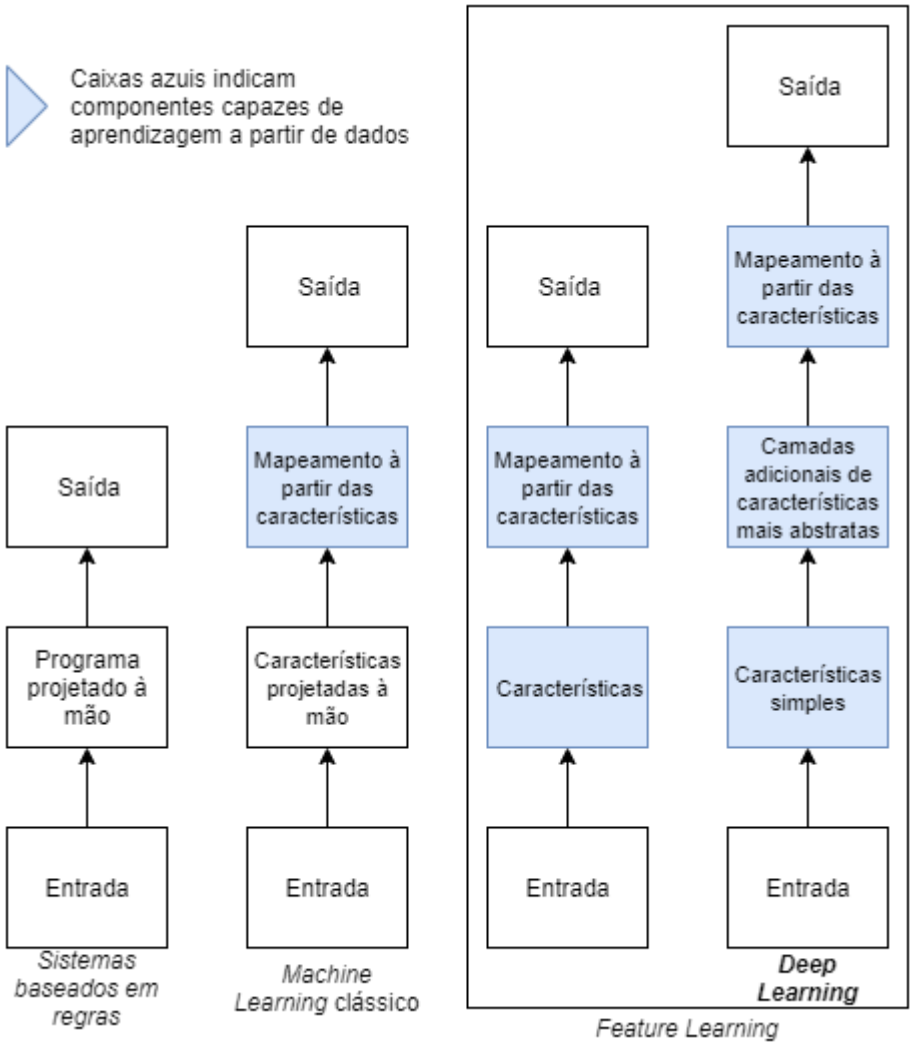


Figura 13 - Relação de sistemas de IA em diferentes disciplinas. (adaptado a partir de GOODFELLOW et al., 2016).

Técnicas de *deep learning* são técnicas de *feature learning* com múltiplos níveis de representação, obtidos por composição de módulos simples que transformam a representação em um nível cada (iniciando na entrada pura) para uma representação em nível maior e um pouco mais abstrato (LECUN et al., 2015). *Deep Learning* alcança isso utilizando redes neurais com múltiplas camadas de rede.

2.3.1. Neural Networks

O cérebro é um computador altamente complexo, não-linear e paralelo, que tem a capacidade de organizar seus constituintes estruturais, conhecidos por

neurônios, de forma a realizar certos processos muito mais rapidamente que o mais rápido computador digital hoje existente (HAYKIN, 2007). No momento do nascimento, um cérebro tem uma grade estrutura e a habilidade de desenvolver suas próprias regras através do que usualmente denominamos “experiência”, que vai sendo acumulada com o tempo (HAYKIN, 2007).

Uma **Rede Neural** (*Neural Network)* é uma máquina que é projetada para modelar a maneira como o cérebro realiza uma tarefa particular ou função de interesse (HAYKIN, 2007). As 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 Figura 14 mostra um exemplo da estrutura de um nó.

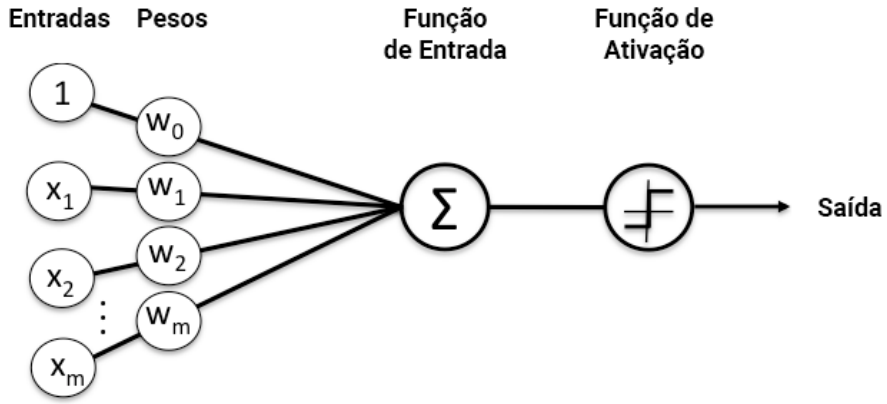


Figura 14 - Diagrama exemplo de um nó (adaptado a partir de PATHMIND, 2019a).

Cada nó realiza uma computação local com a entrada recebida através de uma função de ativação, que acabará gerando um valor numérico de saída (RUSSEL; NORVIG, 2009). Os pesos das ligações entre os nós de uma rede são o método primário de memória a longo termo da rede, com o processo de aprendizado geralmente se tratando da atualização destes valores (RUSSEL; NORVIG, 2009).

As Redes Neurais são geralmente divididas em camadas, no modelo mais básico de rede, temos uma camada de entrada de nós de fonte que se projeta sobre camadas ocultas intermediárias até uma camada de saída (HAYKIN, 2007), como mostra a Figura 15. Existem redes ainda mais simples, com apenas as camadas de entrada e saída.

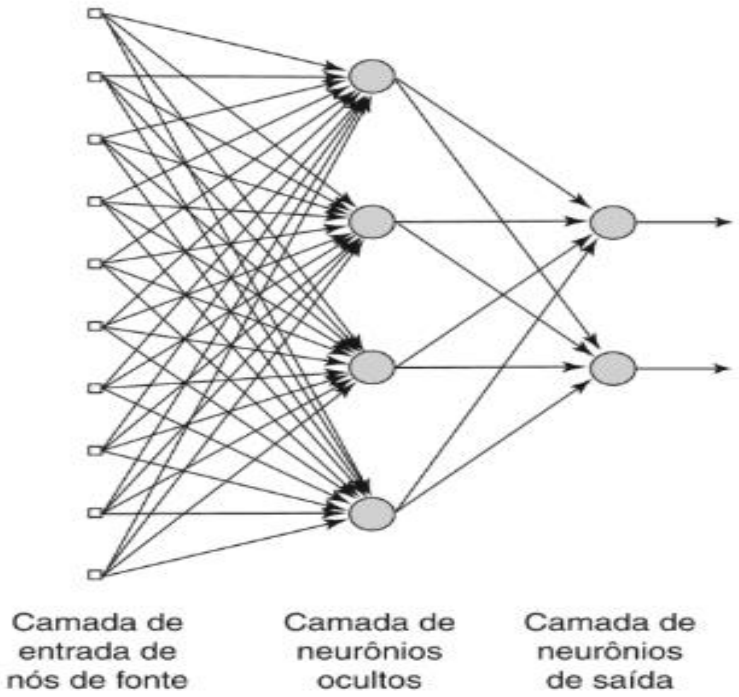


Figura 15 - Estrutura básica de uma rede neural (HAYKIN, 2007).

Uma camada nada mais é do que um conjunto de nós, alimentados pela saída dos nós da camada anterior. É através da propagação de sinais partindo dos nós da camada de entrada, pelos nós ocultos intermediários, até os nós da camada de saída, que a rede realiza sua computação.

2.3.2. *Deep Neural Networks (DNN)*

Redes Neurais Profundas *(Deep Neural Networks)* é a estrutura de rede utilizada em *Deep Learning*. Estas redes se distinguem por sua profundidade, ou seja, o número de camadas pelo qual o dado deve passar em um processo com múltiplas etapas de reconhecimento (PATHMIND, 2019a).

Em redes de *deep learning*, cada camada de nós treina em um conjunto distinto de características baseado na saída da camada anterior. Quanto mais se avança na rede neural, mais complexas as características reconhecidas pelos nós, devido a agregação e combinação de características da camada anterior (PATHMIND, 2019a).

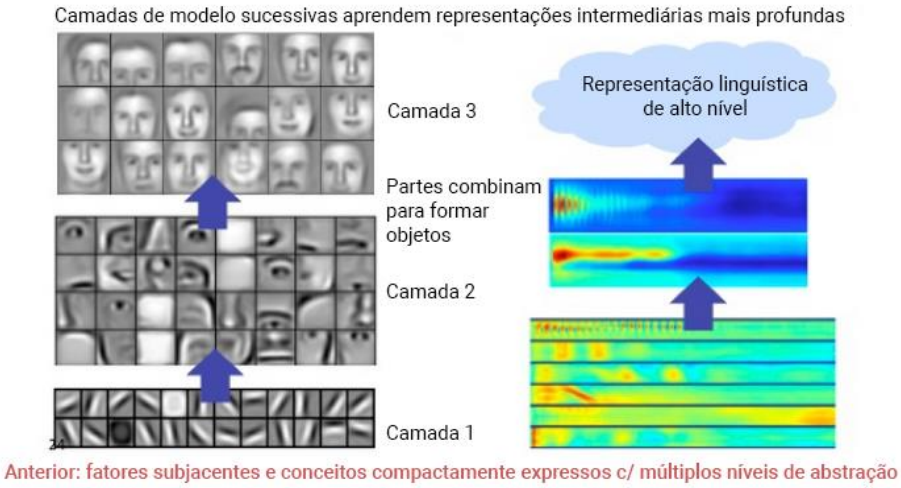


Figura 16 - Hierarquia de características, com aumento da complexidade e abstração a cada nível (adaptado a partir de PATHMIND, 2019a).

Isso é conhecido como hierarquia de características, e é uma hierarquia de complexidade e abstração incrementais, e faz com que redes de *deep learning* sejam capazes de lidar com grandes conjuntos de dados de alta dimensão com bilhões de parâmetros (PATHMIND, 2019a).

Assim, estas redes são capazes de descobrir estruturas latentes em dados não-estruturados e não-categorizados, o que caracteriza a grande maioria dos dados no mundo real (PATHMIND, 2019a). Estes dados também são conhecidos como mídia crua, como fotos, texto, vídeo e gravações de áudio (PATHMIND, 2019a). Portanto, um dos problemas solucionados de melhor maneira por *deep learning* é o processamento e agrupamento da mídia crua e não-categorizada do mundo, detectando similaridades e anomalias em dados que nenhum humano organizou em um banco de dados relacional ou até mesmo nomeou (PATHMIND, 2019a).

2.3.3. *Convolutional Neural Networks (CNN)*

Redes Neurais Convolucionais *(Convolutional Neural Networks)* são redes neurais primariamente utilizadas para classificação/agrupamento de imagens e reconhecimento de objetos. São capazes de reconhecer rostos, indivíduos, sinais de trânsito, animais e diversos outros aspectos de mídia visual (PATHMIND, 2019b).

A eficácia de redes neurais convolucionais em reconhecimento de imagem é um dos principais fatores para o reconhecimento recente da eficácia do *deep learning*. Hoje, *CNNs* são o centro de grandes avanços na área de visão computacional, com uso em carros autônomos, robótica, *drones*, segurança, diagnósticos médicos e tratamento para deficientes visuais (PATHMIND, 2019b).

O modelo de redes neurais convolucionais engloba diversas arquiteturas, que variam em função de ativação dos nós, tipos de camadas etc. A Figura 17 mostra um esquema com comparação de diversos modelos utilizados para visão computacional.

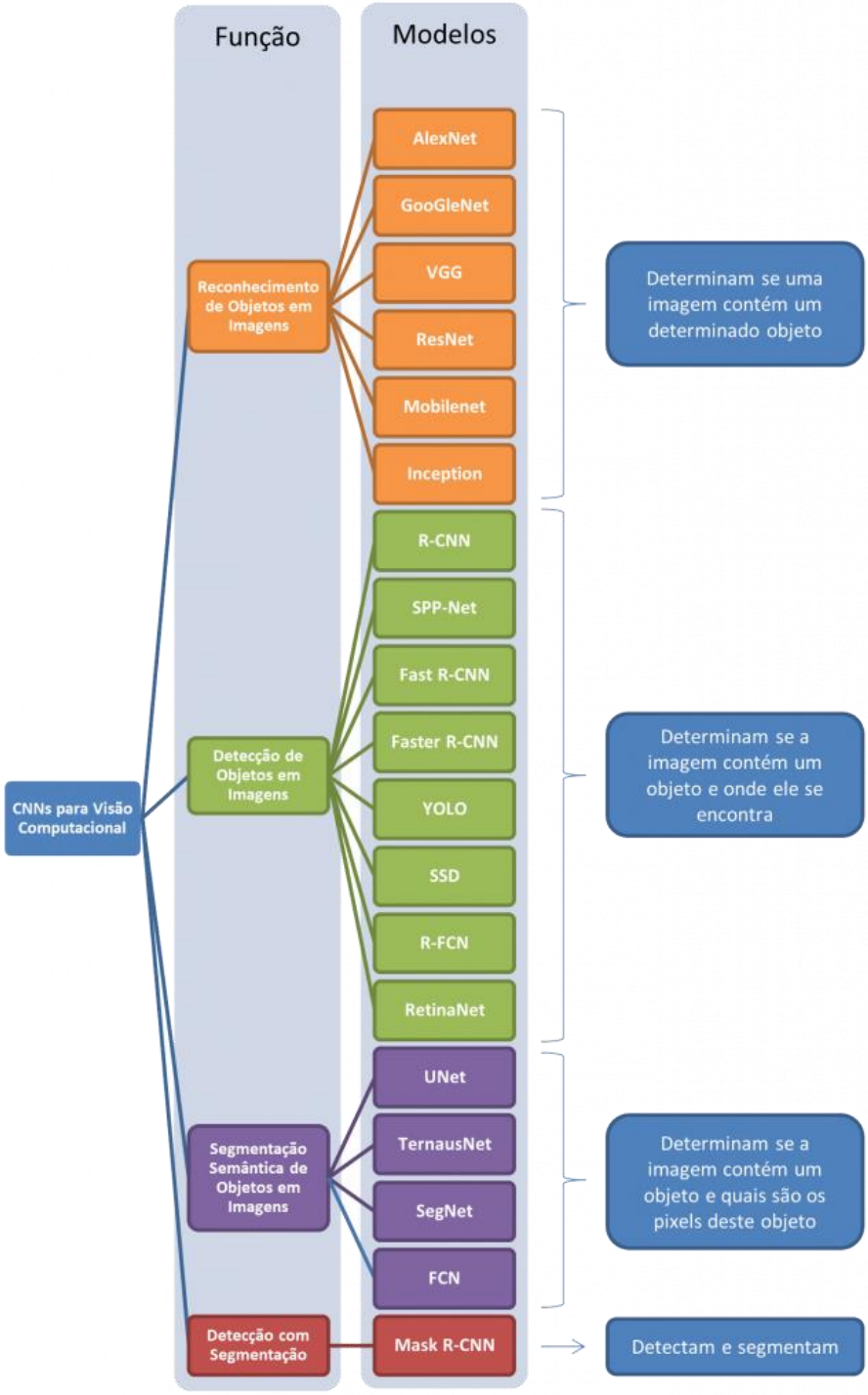


Figura 17 - Redes Neurais Convolucionais para Visão Computacional (VON WANGENHEIM, 2018)

Como o nome “redes neurais convolucionais” indica, esta classe de redes faz uso da operação matemática **convolução**. Redes neurais convolucionais são simplesmente redes neurais que usam a convolução no lugar de multiplicação de matrizes em ao menos uma de suas camadas (GOODFELLOW et al., 2016).

Tradicionalmente, o processo de convolução é uma integral que expressa a sobreposição de uma função *g* ao ser deslocada sobre uma outra função *f*, assim “combinando” as duas funções (WEISSTEIN, 2019). De modo abstrato, uma convolução pode ser definida como o produto de duas funções *f* e *g* (WEISSTEIN, 2019), como mostra a Figura 18.

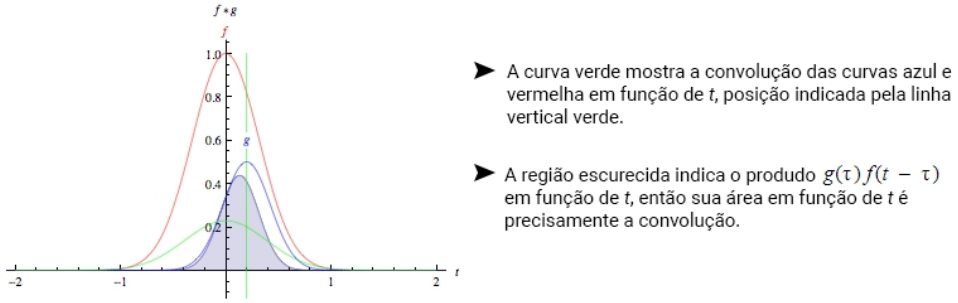


Figura 18 - Convolução (adaptado a partir de WEISSTEIN (2019)).

Para a análise de imagens, a função estática *f* nada mais é do que a imagem de entrada a ser analisada, enquanto a função móvel *g* é conhecida como o filtro, pois detecta sinais ou características da imagem (PATHMIND, 2019b). Em uma *CNN*, geralmente são utilizados diversas sequencias de filtros diferentes para mapear as ocorrências de diferentes características da imagem (PATHMIND, 2019b).

Este tipo de rede neural enxerga imagens como volumes, ou seja, objetos tridimensionais. Isto ocorre devido a imagens digitais serem geralmente representadas utilizando uma codificação *Red-Blue-Green (RGB)*, com seus canais de cores sendo armazenados separadamente, e misturados para produzir o espectro de cores perceptível pelo olho humano. Cada canal de cor é representado por uma matriz de valores, que representam a intensidade do canal em questão em um determinado pixel (PATHMIND, 2019b). A Figura 19 mostra uma aproximação da representação de um canal de cor em uma imagem.

Figura 19 - Comparação de uma imagem com sua representação matricial (adaptado a partir de KARKARE (2019)).

O filtro utilizado para convolução é uma pequena matriz de valores, que percorre a imagem um pixel por vez, multiplicando seus valores com os valores da imagem, somando-os em um único valor para cada sobreposição (KARKARE, 2019), como mostrado na Figura 20.

Figura 20 - Exemplo de Convolução (adaptado a partir de KARKARE, 2019).

A aplicação de convolução permite realçar certas características da imagem, como a presença de linhas horizontais e verticais (KARKARE, 2019). A Figura 21 mostra um exemplo disso.

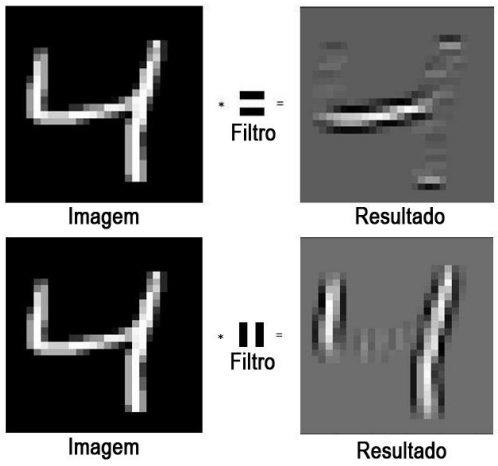


Figura 21 - Linhas verticais e horizontais realçadas através da convolução (adaptado a partir de KARKARE (2019)).

Após uma camada convolucional, é comum adicionar em CNNs uma camada de *pooling,* também conhecido como *downsampling* ou *subsampling*. Esta camada tem como objetivo reduzir as dimensões dos dados, de modo a reduzir o número de parâmetros e computação na rede (KARKARE, 2019). O tipo de *pooling* mais utilizado é o *max pooling*, onde se divide o dado em janelas, mantendo apenas o valor máximo destas, reduzindo o tamanho do dado, mas mantendo as informações significantes (KARKARE, 2019). A Figura 22 mostra um exemplo de *max pooling*.

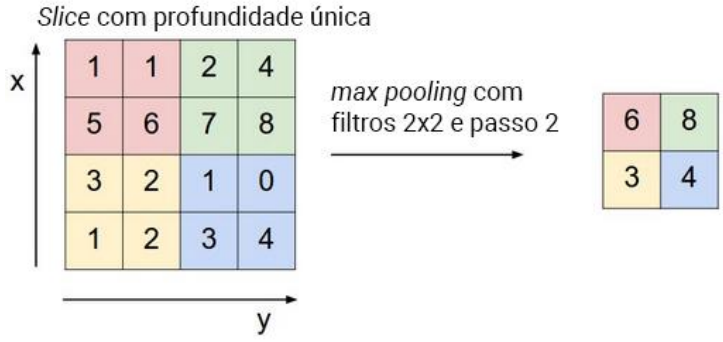


Figura 22 - *Max Pooling* (adaptado a partir de KARKARE, 2019).

A estrutura padrão de uma CNN se dá pela alternação de camadas de convolução e *pooling*, como mostra a Figura 23.

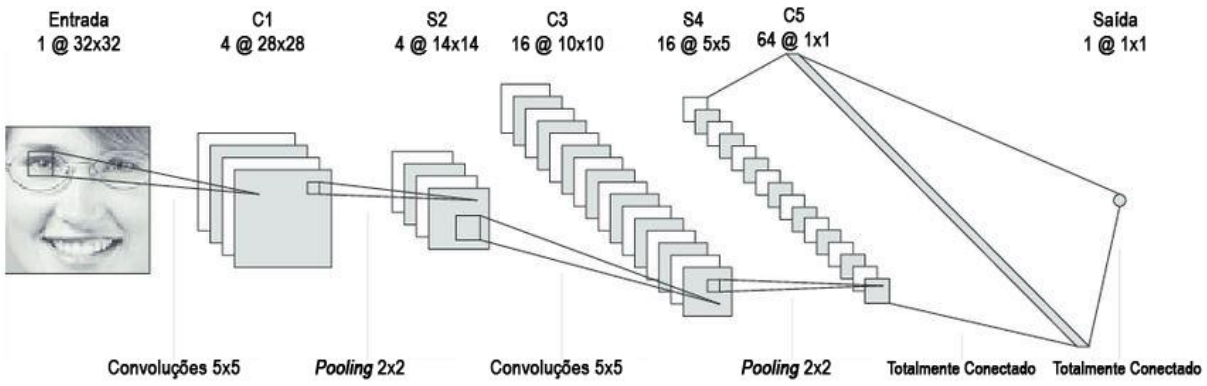


Figura 23 - Arquitetura exemplo de uma CNN (adaptado a partir de KARKARE, 2019).

Por meio desta combinação de convolução com *pooling*, é possível extrair diferentes características da imagem, com menor custo computacional.

2.3.4. Treinando uma Rede Neural

O modelo de *machine learning* não passa de um algoritmo, que se torna inteligente após ser treinado com um conjunto de dados. Logo, se o modelo receber dados inadequados, retornará resultados inadequados (PANT, 2019).

O processo de treinamento de uma rede inicia com o agrupamento de dados, sejam eles numéricos, categóricos ou ordinais. Este agrupamento geralmente ocorre pela utilização de conjuntos de dados disponibilizados por terceiros, criação de um conjunto de dados próprio ou pela junção de ambos (PANT, 2019).

Uma vez coletados, os dados devem passar por um pré-processamento, onde os dados coletados no mundo real são convertidos em um conjunto de dados limpo. Isso é necessário pois dados do mundo real podem apresentar alguns problemas, por exemplo:

- **Dados ausentes:** Dados podem estar incompletos quando não gerados continuamente ou por problemas técnicos na aplicação.

- **Dados com ruído:** Também conhecidos como *outliers*, estes dados podem ocorrer devido a erro humano, ou algum problema técnico durante a coleta de dados.
- **Dados inconsistentes:** Estes dados podem ocorrer devido a erro humano ou duplicação de dados (PANT, 2019).

Alguns exemplos de técnicas de pré-processamento para conversão em dados limpos são:

- **Conversão de dados:** Modelos de *machine learning* são capazes de lidar apenas com características numéricas, então dados categóricos e ordinais devem ser convertidos para valores numéricos.
- **Ignorar dados ausentes:** Quando é detectada a ausência de dados em um conjunto de dados, a linha ou coluna de dados pode ser removida dependendo da necessidade.
- **Completar dados ausentes:** Quando é detectada a ausência de dados em um conjunto de dados, este pode ser completado manualmente, por meio da média, mediana ou moda dos demais dados.
- ***Machine Learning*:** Se é detectada a ausência de dados, podem ser utilizadas técnicas de *machine learning* em uma tentativa de adivinhar os dados ausentes com base nos demais dados.
- **Detecção de *outliers*:** Pode ocorrer a presença de erros no conjunto de dados que desvia drasticamente de outras observações do conjunto, que podem ser eliminados (PANT, 2019).

Uma vez que o conjunto de dados limpos está disponível, deve ser determinado os modelos de rede e treinamento mais adequados para o problema em questão (PANT, 2019). A Figura 24 mostra alguns exemplos de modelos de treinamento.

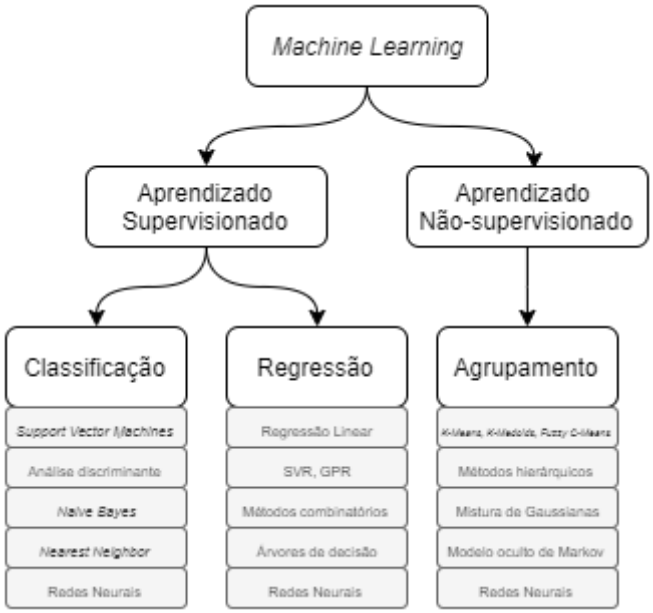


Figura 24 - Overview dos modelos de treinamento (adaptado a partir de PANT, 2019).

No **treinamento supervisionado**, um sistema de *machine learning* recebe dados previamente categorizados de maneira correta, e é dividido em duas categorias: Classificação e Regressão (PANT, 2019). Em problemas de **classificação**, a variável alvo é categórica, ou seja, os dados são classificados na classe A, classe B, ou alguma outra classe. Em problemas de **regressão**, a variável alvo é contínua (numérica) (PANT, 2019).

Já no **treinamento não-supervisionado**, o sistema recebe apenas os dados, sem categorização prévia, e cabe a ele determinar padrões entre estes. Este tipo de treinamento geralmente é utilizado para problemas de **agrupamento**, onde o objetivo é dividir os dados em grupos, não conhecidos previamente (PANT, 2019).

Durante a criação de uma rede neural, os pesos de suas ligações geralmente recebem valores aleatórios, sendo atualizados ao longo do processo de treinamento

por algum algoritmo, de modo a encontrar os valores ideais. Para se determinar quais valores são melhores que outros, é necessário determinar quão próximo o resultado obtido está do valor esperado, que é geralmente feito por meio de uma função de perda, utilizada devido a sua maior facilidade de minimização quando comparada à maximização da acurácia (BUSHAEV, 2017).

O problema de treinamento é então equivalente ao problema de minimização da função perda, o que pode ser feito utilizando diversos algoritmos, com gradiente descendente estocástico sendo o mais utilizado (BUSHAEV, 2017). Assim, ao longo do processo de treinamento, a rede calcula o valor da função perda, e ajusta os pesos de suas ligações de acordo com o algoritmo escolhido, até que o erro seja minimizado ou o conjunto de dados para treinamento termine.

2.3.5. *Frameworks* e Bibliotecas

Frameworks e bibliotecas são igualmente código disponibilizado para ser utilizado por terceiros para resolver problemas comuns, eliminando a necessidade de criação de código repetido. Bibliotecas são invocadas por programadores em seu código, enquanto um *framework* já possui todo fluxo de controle, restando ao programador apenas o preenchimento das lacunas restantes. A Tabela 2 traz uma lista dos *frameworks* de redes neurais predominantes hoje.

Tabela 5 - Frameworks de Redes Neurais predominantes

Framework	**Desenvolvido por**	**Principais linguagens compatíveis**
Tensorflow	Google Brain Team	C++, Python
The Microsoft Cognitive Toolkit (CNTK)	Microsoft	C++, C#, Python, Java
Torch	Ronan Collobert, Koray Kavukcuoglu, Clement Farabet	C, LuaJIT
PyTorch	Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan	C++, Python
Keras	François Chollet	Python
Caffe	Berkeley AI Research (BAIR)	C++, Python
Eclipse Deeplearning4j	Pathmind	Java, Scala, Clojure, Kotlin

Chainer	Preferred Network em parceria com IBM, Intel, Microsoft e Nvidia	Python
Darknet	Joseph Redmon	C++, Python

Cada *framework* possui suas vantagens e desvantagens, tornando-os opções viáveis dependendo de sua aplicação. Contudo, *frameworks* requerem um esforço maior por parte do usuário para entender e fazer uso de suas funções quando comparados a bibliotecas. Para solucionar isso, foram desenvolvidas bibliotecas com base nesses *frameworks*, visando aumentar o nível de abstração e assim facilitar o seu uso e tornar o poder do estado-da-arte em *deep learning* mais acessível e disponível para qualquer pessoa (Fastai, 2019).

Uma das bibliotecas predominantes é a biblioteca **fast.ai**. Ela utiliza do *framework* PyTorch v1 e provê uma única e consistente *Application Programming Interface* (API) para as aplicações de *deep learning* e tipos de dados mais importantes (HOWARD, 2018).

O *framework* Darknet é utilizado principalmente para o desenvolvimento de modelos de detecção de objetos YOLO (*You Only Look Once*), uma nova abordagem para detecção de objetos, capaz de detectar *bounding boxes* de objetos com performance equivalente ou superior a outros modelos equivalentes (REDMON et al., 2015).

3. ESTADO DA ARTE

A fim de levantar o estado da arte sobre a geração automática de *wireframes* a partir de *sketches*, são apresentados neste capítulo os resultados de um mapeamento sistemático da literatura com base no procedimento definido por Petersen et al. (2008).

3.1. Definição do protocolo de revisão

O Objetivo da realização deste mapeamento sistemático é responder a seguinte pergunta de pesquisa: Quais abordagens existem para a **geração automática de *wireframes* de interface a partir de *sketches* usando *Machine Learning***?

Para realização desse mapeamento, esta pesquisa foi refinada nas seguintes perguntas de análise:

PA1. Quais modelos/ferramentas para geração de *wireframes* a partir de *sketches* existem?

PA2. Quais os dados de entrada/saída?

PA3. Quais elementos são detectados e de qual tipo de software?

PA4. Quais conjuntos de dados usaram e quais as características desses conjuntos?

PA5. Qual é tipo de rede que usaram e quais as características dessa rede?

PA6. Como foi medida a qualidade do resultado e quais resultados foram obtidos?

Fontes: Foram escolhidas para a realização da busca as principais bases de dados e bibliotecas digitais do campo da computação, incluindo ACM Digital Library, IEEE Xplore Digital Library, arXiv.org E-print Archive e Scopus com acesso via portal Capes.

***String* de busca:** Se baseando na pergunta de pesquisa, para calibração da *string* de busca foram realizadas diversas buscas informais, com termos de buscas relevantes e seus sinônimos (Tabela 6). Usou se também sinônimos para minimizar o risco de omissão de trabalhos relevantes.

Tabela 6 - Termos de busca e respectivos sinônimos

Termo de Busca	**Sinônimo(s)**
sketch	*sketches, mockup, mockups, screenshot, screenshots*
wireframe	*wireframes*
user interface	*user interfaces, ui*
machine learning	*deep learning, neural network, neural networks, cnn, computer vision*

Após a realização das buscas informais, definiu-se uma *string* de busca específica, para aplicar nas bases de dados mencionadas de modo a encontrar todos os artigos relevantes previamente conhecidos, e um número satisfatório de artigos possivelmente relevantes adicionais:

(sketch OR wireframe* OR screenshot* OR mockup*) AND (ui OR "user interface*") AND (app* OR website* OR iOS OR mobile OR Android) AND (“machine learning" OR "deep learning" OR "neural network*" OR cnn OR "computer vision”)*

Uma vez definida a *string* de busca, foi realizada sua adaptação para as diferentes bases de dados consideradas (Tabela 7).

Tabela 7 - *Strings* de buscas utilizadas nas diferentes bases de dados

Base de Dados	***String* de Busca**
ACM Digital Library	(sketch* OR wireframe* OR screenshot* OR mockup*) AND (ui OR "user interface*") AND (app* OR website* OR ios OR mobile OR android) AND ("machine learning" OR "deep learning" OR "neural network*" OR cnn OR "computer vision")
IEEE Xplore Digital Library	(sketch* OR wireframe* OR screenshot* OR mockup*) AND (ui OR "user interface" OR "user interfaces") AND (app* OR website OR websites OR ios OR mobile OR android) AND ("machine learning" OR "deep learning" OR "neural network" OR "neural networks" OR cnn OR "computer vision")

Scopus	TITLE-ABS-KEY ((sketch* OR wireframe* OR screenshot* OR mockup*) AND (ui OR "user interface*") AND (app* OR website* OR ios OR mobile OR android) AND ("machine learning" OR "deep learning" OR "neural network*" OR cnn OR "computer vision")) AND PUBYEAR > 2009
arXiv.org e-print archive	order: -announced_date_first; size: 50; date_range: from 2009-01-01 ; include_cross_list: True; terms: AND all=sketch* OR wireframe* OR screenshot* OR mockup*; AND all=ui OR "user interface*"; AND all=app* OR website* OR ios OR mobile OR android; AND all="machine learning" OR "deep learning" OR "neural network*" OR cnn OR "computer vision"

Onde possível, os resultados foram limitados a artigos publicados na última década (desde 2009), devido à natureza recente de alguns conceitos relevantes, como aplicativos moveis.

Critérios de inclusão e exclusão

- São considerados artigos científicos e artefatos que apresentem modelos de geração de *wireframes* ou reconhecimento de *sketches* de interface.
- São incluídos apenas artefatos em inglês.
- Foram considerados apenas artigos que apresentem geração automática de um modelo a partir de alguma outra representação de interface de usuário.
- Para tornar a pesquisa mais ampla, foram incluídos também artefatos que utilizem capturas de tela ou *sketches* feito por uma ferramenta de software no lugar de *sketches* manuais.
- Para tornar a pesquisa mais ampla, foram incluídos também artefatos que gerem uma saída diferente de *wireframes* em App Inventor, incluindo representações intermediarias da estrutura da tela, códigos etc.

Critérios de qualidade: Foram considerados apenas artigos que apresentem informação substancial sobre a abordagem da geração automática.

3.2. Execução da Busca

A busca dos artigos foi realizada em setembro e outubro de 2019 pelo autor do presente trabalho e revisado pela orientadora. A busca inicial resultou em 88 artigos, dos quais foram selecionados os artigos relevantes de acordo com os critérios de inclusão, exclusão e qualidade (Tabela 8).

Tabela 8 - Resultados da Busca

Base de Dados	***Quantidade de artigos resultantes da busca***	***Quantidade de artigos relevantes***
ACM Digital Librar	34	5
IEEE Xplore Digital Library	17	1
Scopus	35	2
arXiv.org e-print archive	2	1
Total	*88*	*9*

Por meio da leitura dos títulos e resumos de todos os trabalhos encontrados na busca inicial, foram determinados os artigos potencialmente relevantes ao tema de acordo com os critérios de inclusão e exclusão, sendo selecionados os artigos realmente relevantes por meio de sua leitura completa e aplicação dos critérios de qualidade.

Como alguns artigos relevantes foram encontrados em mais de uma base de dados, se escolheu apenas uma destas para selecionar o artigo. Após a aplicação de todos os critérios, foram identificados no total 9 trabalhos relevantes publicados (Tabela 8).

3.3. Análise dos Resultados

Para responder à questão de pesquisa, as informações relevantes às perguntas de análise foram extraídas dos 9 artigos relevantes encontrados conforme especificado na Tabela 9.

Tabela 9 - Especificação das Informações Extraídas.

Pergunta de análise	Dados a extrair	Descrição
PA1. Quais modelos/ferramentas existem?	Nome	O nome ou o autor da UI
	Referência	Referência bibliográfica
PA2. Quais os dados de entrada/saída?	Dados de Entrada	O tipo de dado de entrada (*Screenshot*, *sketch* etc.)
	Formato dos dados de Entrada	Formato dos arquivos de entrada
	Dados de Saída	O tipo de dado de saída (Árvore, *wireframe*, código etc.)
	Formato dos dados de Saída	Formato dos arquivos de saída
PA3. Quais elementos de interface de usuário foram detectados e de qual tipo de software?	Elementos detectados	Elementos de Interface detectados (Botões, imagens etc.)
	Tipo de software	Tipo de Software das UIs (Web, Apps etc.)
	Plataforma	Android, iOS, PC etc.
PA3. Quais conjuntos de dados usaram e quais as características desses conjuntos?	Nome do Dataset	Nome do *Dataset* utilizado
	Descrição	Descrição do *Dataset*
	Labeling	Formato dos *labels* e como foi realizado o processo de *labeling*
	Pré-processamento	Pré-processamento realizado nos dados
	Formato	Formato dos dados no *dataset*
	Quantidade de instancias	O número de instancias de dados presente no *dataset*.
	Licença	Licença do *dataset*
	Data de criação	Data de criação do *dataset*
	Referência	Referência para o *dataset*
PA4. Qual tipo de rede usaram e quais as características dessa rede?	Qual o modelo de rede utilizado?	Detalhes sobre a Rede utilizada
	Tipo de aprendizagem	Tipo de aprendizagem (supervisionada, não-supervisionada etc.)
PA5. Como foi medida a qualidade do resultado e quais resultados foram obtidos?	Medidas de desempenho	Medidas utilizados para avaliar o desempenho do modelo acurácia, taxas de positivos verdadeiros e falsos, precisão etc.
	Avaliação do modelo	Avaliação do modelo e comparação com demais modelos, por teste de hipótese, análise de correspondência, correlação etc.
	Resultados da avaliação:	Descrição dos principais resultados, pontos positivos e negativos identificados na avaliação do modelo
PA6. Qual a forma de implantação da rede/modelo de *machine learning*?	Descrição de como o modelo é implantado para uso.	Existe algum sistema disponível ou planos para sua implantação?

Os artigos selecionados foram lidos de forma completa e os dados foram extraídos pelo autor revisado 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. Quais modelos/ferramentas existem?

No total foram encontrados 9 artefatos artigos que apresentam alguma forma de geração automática de *wireframes* de interface a partir de *sketches* usando *Machine Learning* (Tabela 10).

Tabela 10 – Documentos resultantes da execução da busca

Citação	Referência Bibliográfica
(BELTRAMELLI, 2018)	BELTRAMELLI, T.; **pix2code: Generating Code from a Graphical User Interface Screenshot**. EICS '18: ACM SIGCHI Symposium on Engineering Interactive Computing Systems, Paris, France. 2018.
(GE, 2019)	GE, X. **Android GUI search using hand-drawn sketches**. In: Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE '19), Piscataway, USA. 2019
(KIM et al., 2018)	KIM, B.; PARK, S.; WON, T.; HEO, J.; KIM, B. 2018. **Deep-Learning Based Web UI Automatic Programming**. In: Proceedings of International Conference on Research in Adaptive and Convergent Systems, Honolulu, USA, 2018.
(NGUYEN et al., 2015)	NGUYEN, T. A.; Christoph CSALLNER, C. **Reverse engineering mobile application user interfaces with REMAUI**. In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, Piscataway, USA, 2015.
(HUANG et al., 2019)	HUANG, F; F. CANNY, J; NICHOLS, J. **Swire: Sketch-based User Interface Retrieval**. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, Glascow, Escócia, 2019.
(ROBINSON, 2019)	ROBINSON, A. **Sketch2code: Generating a website from a paper mockup**, 2019.
(LIU et al., 2018)	LIU, Y.; HU, Q.; SHU, K.; **Improving pix2code based Bi-directional LSTM**. In: IEEE International Conference on Automation, Electronics and Electrical Engineering (AUTEEE), Shenyang, China, 2018.
(YUN et al., 2018)	YUN, Y; JUNG, J.; EUN, S.; SO, S.; HEO, J. **Detection of GUI elements on sketch images using object detector based on deep neural networks**. 6th International Conference on Green and Human Information Technology, Chiang Mai, Thailand, 2018
(MORAN et al., 2018)	MORAN, K. P.; BERNAL-CARDENAS, C.; CURCIO, M.; BONETT, R.; POSHYVANYK, D. **Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps**. In: in IEEE Transactions on Software Engineering, 2018.

Pode se observar que a preocupação com a geração automática de *wireframes* a partir de *sketches* de interfaces de usuário é recente, com o relato relevante mais antigo encontrado durante a busca sendo publicado em 2015, e um aumento significante nos dois últimos anos (Figura 25).

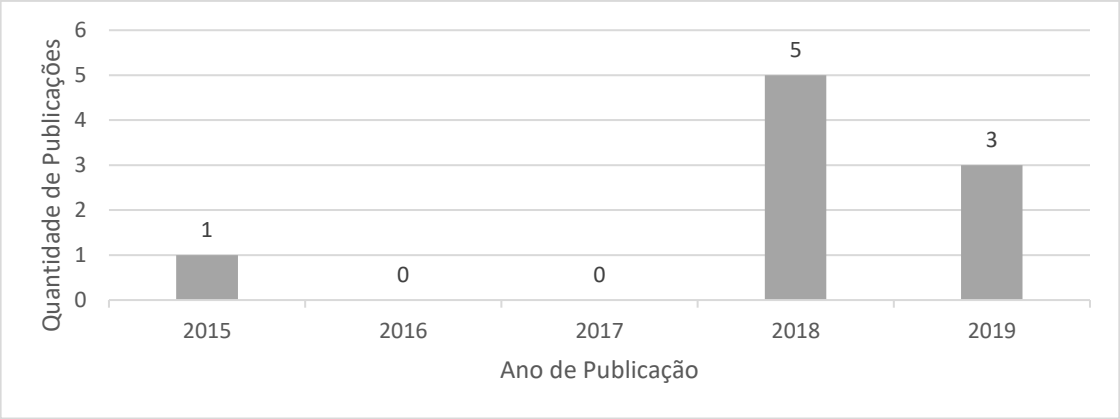


Figura 25 - Quantidade de publicações relevantes resultantes da busca

PA2. Quais os dados de entrada/saída?

Analisando os artefatos resultantes da busca, foi possível observar grande variação nos dados, principalmente de saída (Tabela 11).

Tabela 11 - Dados extraídos para responder à Pergunta de Análise 2

Citação	**Dados**			
	Dados de Entrada	**Formato dos dados de Entrada**	**Dados de Saída**	**Formato dos dados de Saída**
(BELTRAMELLI, 2018)	Screenshots	Imagens	Código	*Domain-Specific Language* Própria
(GE, 2019)	Sketches	Imagens	DL Framework gera um esqueleto da GUI (JSON), mas a saída final são APKs com GUIs similares	JSON/APKs
(KIM et al., 2018)	Sketch	Imagens	Código da UI	HTML
(NGUYEN et al., 2015)	Screenshots e Sketches	Bitmaps	Projeto de Android Completo	Pasta de Projeto
(HUANG et al., 2019)	Sketches	Imagens	Uis similares	Screenshots
(ROBINSON, 2019)	Sketches	Fotos	UI	HTML
(LIU et al., 2018)	Screenshots	NI (Imagens)	Código	*Domain-Specific Language* Própria
(YUN et al., 2018)	Sketches	Imagens	Código	XML
(MORAN et al., 2018)	Computer Mockups	Imagens	Código	XML

Dentre os resultados encontrados, é possível ver uma divisão entre principalmente dois tipos de dados de entrada distintos, *Sketches* ou *Screenshots* de interface (Figura 26), com apenas MORAN et al. (2018) partindo de *Mockups* feitas em computador.

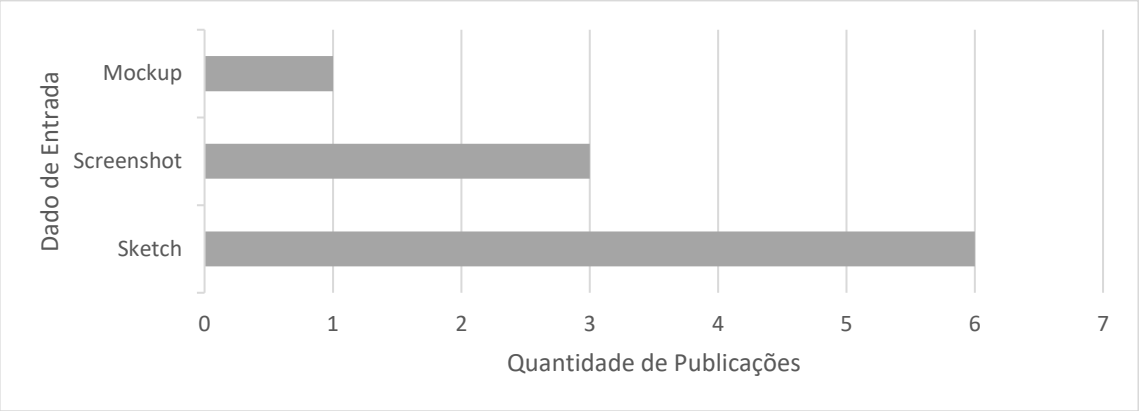


Figura 26 - Tipo de Dado de Entrada por quantidade de publicações

O formato dos dados de entrada é especificado em poucas publicações, com apenas Nguyen et al. (2015) especificando a utilização de *Bitmaps*, enquanto os demais artigos se limitam ao termo imagem.

Quanto aos dados de saída, todos os artigos resultantes da busca geram alguma representação da UI em código, exceto Huang et al. (2019), que retorna apenas *screenshots* de UIs semelhantes.

Beltramelli (2018) e Liu et al. (2018) utilizam linguagens de domínio específico próprias para representação da UI de saída, enquanto Kim et al. (2018) e Robinson et al. (2019) utilizam HTML e Yun et al. (2018) e Moran et al. (2018) utilizam XML. Nguyen et al. (2015) se destaca por fornecer como saída projetos de Android Studio completos.

Ge (2019) se diferencia dos demais artigos por gerar em um passo intermediário uma representação da interface em JSON, enquanto sua saída ao final são APKs de aplicativos com interfaces visualmente similares.

PA3. Quais elementos de interface de usuário foram detectados e de qual tipo de software?

Dentre os artefatos analisados, quando especificados os elementos de UI detectados pelo sistema, ocorre geralmente um foco apenas nos mais populares (Tabela 12).

Tabela 12 - Dados extraídos para responder à Pergunta de Análise 3

Citação	Dados		
	Elementos detectados	Tipo de software	Plataforma
(BELTRAMELLI, 2018)	NI	Diversos	IoS, Android e Web
(GE, 2019)	TextView, EditText, ImageView, Button, RadioButton, Switch and CheckBox	Aplicativos	Android
(KIM et al., 2018)	Button, RadioButton, checkBox, editText, text, etc.	Web	Web
(NGUYEN et al., 2015)	Layout (LinearLayout, RelativeLayout, FrameLayour) e Widgets (TextView, ImageView e Button)	Aplicativos	Android
(HUANG et al., 2019)	NI	Aplicativos	Android
(ROBINSON, 2019)	Title, Image, Button, Input e Paragraph	Web	Web
(LIU et al., 2018)	NI	Diversos	IoS, Android e Web
(YUN et al., 2018)	NI	Apps	NI
(MORAN et al., 2018)	Os 15 mais Populares (TextView, ImageView, Button, ImageButton, EditText, CheckedTextView, CheckBox, RadioButton, ProgressBar, SeekBar, NumberPicker, Switch, ToggleButton, RatingBar e Spinner)	Apps	Mobile

Moran et al. (2018) entra em grandes detalhes sobre a análise feita para determinar os elementos mais utilizados dentre os aplicativos mais populares, resultando em seu *dataset* próprio com a contagem de elementos mostrada em uma tabela (Figura 27).

GUI-C Type	Total # (C)	Tr (O)	Tr (O+S)	Valid	Test
TextView	99,200	74,087	74,087	15,236	9,877
ImageView	53,324	39,983	39,983	7,996	5,345
Button	16,007	12,007	12,007	2,400	1,600
ImageButton	8,693	6,521	6,521	1,306	866
EditText	5,643	4,230	5,000	846	567
CheckedTextView	3,424	2,582	5,000	505	337
CheckBox	1,650	1,238	5,000	247	165
RadioButton	1,293	970	5,000	194	129
ProgressBar	406	307	5,000	60	39
SeekBar	405	304	5,000	61	40
NumberPicker	378	283	5,000	57	38
Switch	373	280	5,000	56	37
ToggleButton	265	199	5,000	40	26
RatingBar	219	164	5,000	33	22
Spinner	20	15	5,000	3	2
Total	191,300	143,170	187,598	29,040	19,090

Figura 27 - Quantidade de cada elemento de interface presentes em interfaces do *dataset* REDRAW (MORAN et al., 2018)

Ge (2019), Kim et al. (2018), Nguyen et al. (2015) e Robinson (2019) seguem caminhos semelhantes, com foco em apenas um certo número de componentes dentre os mais populares, uma vez que estes representam uma proporção muito maior dos casos encontrados.

Apenas Nguyen et al. (2015) especifica a detecção de elementos de *layout* como *LinearLayout*, *RelativeLayout* e *FrameLayout.*

Também é possível observar que dentre os artefatos encontrados um foco em interfaces de aplicativos mobile (Figura 28), com 7 publicações sendo voltadas a este tipo de software, enquanto apenas 4 publicações têm foco em softwares web. Beltramelli (2018) e Liu et al. (2018) são os únicos artefatos encontrados que apresentam abordagens com foco em ambos.

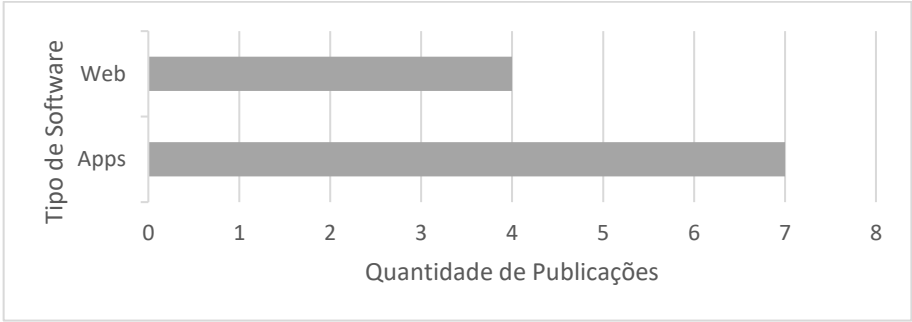


Figura 28 – Quantidade de Publicações por Tipo de Software

Dentre os artefatos com foco em aplicativos mobile, é possível ainda observar maior interesse em aplicativos da plataforma Android, com 5 publicações (Figura 29).

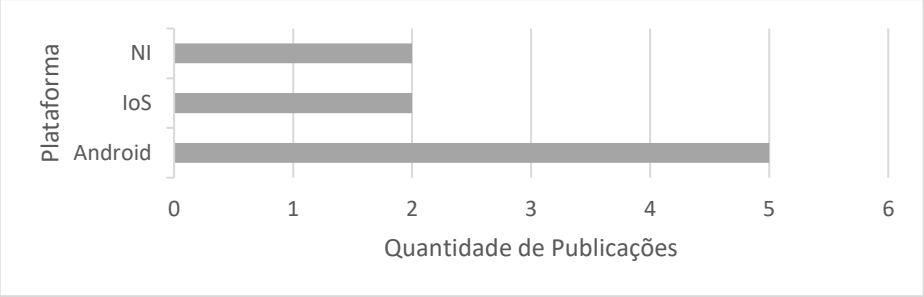


Figura 29 - Quantidade de Publicações por Plataforma de Apps

A plataforma IoS é mencionada apenas nas abordagens apresentadas por Beltramelli (2018) e Liu et al. (2018), enquanto Yun et al. (2018) e Moran et al. (2018) não entram em detalhes quanto a plataforma móvel utilizada.

PA4. Quais conjuntos de dados usaram e quais as características desses conjuntos?

Analisando os artigos relevantes, foi possível perceber a ausência de um *dataset* conhecido com imagens de *sketches* de interfaces, com muitos dos artigos criando seu próprio *dataset* (Tabela 13).

Tabela 13 - Dados extraídos para responder à Pergunta de Análise 4

Citação	Dados								
	Nome do *Dataset*	**Descrição**	***Labeling***	**Pré-processamento**	**Formato de arquivo**	**Quantidade de instancias**	**Licença**	**Data de criação**	**Referência**
(BELTRAMELLI, 2018)	Pix2code Dataset	*Screenshots* de interfaces geradas automaticamente	Cada Screenshot tem um arquivo .gui correspondente	NI	.png + .gui	5.250 (1.750 Android, 1.750 iOs, 1.750 Web)	APACHE2 - Disponível no github	Setembro de 2017	https://github.com/tonybeltramelli/pix2code/tree/master/datasets
(GE, 2019)	Rico	Dataset com design de Uis de Apps	NI	Dataset convertido em sketches gerados automaticamente	Screenshots, uma visualização da hierarquia Android aumentada, um conjunto de interações de usuário, um conjunto de animações com transições, e uma representação em vetor da hierarquia	72k	NI	NI	NI
(KIM et al., 2018)	NI	NI	NI	NI	NI	NI	NI	NI	NI
(NGUYEN et al., 2015)	NI	NI	NI	NI	NI	NI	NI	NI	NI
(HUANG et al., 2019)	Swire Dataset	Sketches de Screenshots retirados da base de dados Rico	NI	NI	jpg (invertido, fundo preto com linhas brancas)	3802 Sketches de 2201 UIs de 167 Apps	NI	Última atualização em janeiro de 2019	https://github.com/huang4fstudio/swire
(ROBINSON, 2019)	NI	Sketches gerados a partir de Websites Bootstrap reais, normalizados e com seus elementos substituídos por modelos de Sketches equivalentes, e seus correspondentes HTMLs normalizados.	Cada pixel com o elemento que representa	Tanto Sketch quanto imagem normalizada redimensionados para 256x256	NI	1750	NI	NI	NI
(LIU et al., 2018)	pix2code (Apenas a parte Web)	Screenshots de interfaces geradas automaticamente	Cada Screenshot tem um arquivo .gui correspondente	NI	.png + .gui	1.750 Web - Não retirado do Artigo, mas sim do github	APACHE2 - Disponível no github	Setembro de 2017	https://github.com/tonybeltramelli/pix2code/tree/master/datasets
(YUN et al., 2018)	NI	50 sketches com aproximadamente 600 elementos, imitando screenshots da internet	Elementos anotados a mão	NI	Imagem + XML	50	NI	NI	NI
(MORAN et al., 2018)	REDRAW	Dataset utilizado para treinar e avaliar as técnicas CNN e KNN utilizadas para o artigo ReDraw	191,300 componentes de interface categorizados	Filtros	NI	14,382 telas	Creative Commons Attribution 4.0 International	Junho, 2017	https://zenodo.org/record/2530277

Beltramelli (2018) utiliza um *dataset* próprio, com *screenshots* e códigos correspondentes de interfaces geradas automaticamente com parâmetros aleatorizados, Liu et al. (2018) utiliza parte deste mesmo *dataset*.

Como existem *datasets* com *screenshots* de Apps disponíveis para o uso, Ge (2019) e Huang et al. (2019) se baseiam em um destes, o *dataset* Rico (REF) de aplicativos mobile. Ambos os trabalhos realizam uma conversão dos *screenshots* disponíveis no *dataset* em *sketches*. Huang et al. (2019), por um lado, contratou designers para a criação de *sketches* se baseando em *screenshots* selecionados do *dataset*, enquanto Ge (2019) realizou a conversão automática, por meio de técnicas de manipulação de imagem.

Yun et al. (2018) também realizou manualmente a geração de *sketches* com base em *screenshots* de aplicativos retirados da internet, mas em escala menor, com apenas 50 *sketches*.

Robinson (2019) teve uma abordagem diferente para geração de *sketches* a partir de dados disponíveis na internet. Neste artigo, foram utilizadas *templates* para *websites* Bootstrap como base, sendo realizada a normalização destas *templates*, através por meio da eliminação de elementos indesejados e simplificação da estrutura, sendo cada elemento restante substituído por um dos diferentes exemplos de elementos desenhados a mão correspondentes, com as devidas modificações em dimensões e escala (Figura 30).

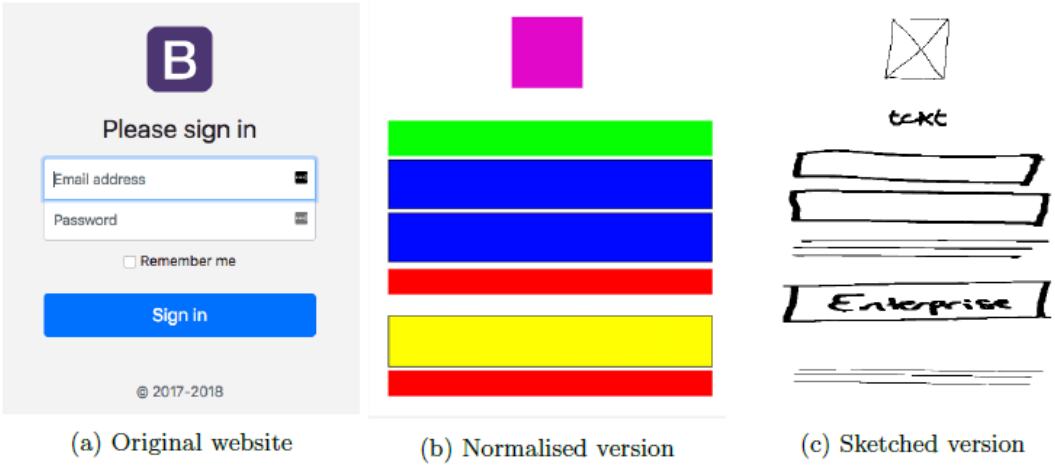


Figura 30 - Conversão de *website* em *sketch* (ROBINSON, 2019)

Moran et al. (2018) também utiliza de interfaces prontas, mas apenas para verificação da proporção de cada tipo de elemento, gerando automaticamente interfaces com proporção similar de cada elemento, de modo a aproximar mais casos de uso real do que a geração de interfaces puramente aleatórias.

Não foi encontrado nenhum *dataset* com screenshots/sketches referente a apps do App Inventor.

PA5. Qual tipo de rede usaram e quais as características dessa rede?

Dentre os artefatos encontrados, é comum a separação do processo em múltiplas etapas, com a utilização de múltiplos modelos de rede neural e/ou técnicas de visão computacional (Tabela 14).

Tabela 14 - Dados extraídos para responder à Pergunta de Análise 5

Citação	**Dados**	
	Modelo de Rede	**Tipo de aprendizagem**
(BELTRAMELLI, 2018)	CNN (*Vision*) -> LSTM (*Language*) -> LSTM (*Decoder*)	Não supervisionado (*vision*), supervisionado (*Decoder*)
(GE, 2019)	*Deep Learning*	NI
(KIM et al., 2018)	*Faster* R-CNN	NI
(NGUYEN et al., 2015)	Visão Computacional	NI
(HUANG et al., 2019)	Duas sub-redes idênticas, similares à *VGG-A deep convolutional neural network*	Supervisionada

(ROBINSON, 2019)	Deeplab v3+ (Xception)	Supervisionada
(LIU et al., 2018)	CNN e Bidirectional LSTM	Supervisionada
(YUN et al., 2018)	DNN + YOLO	Supervisionada
(MORAN et al., 2018)	CV (Detecção de Elementos) -> CNN (Classificação) -> KNN (Geração da Hierarquia)	Supervisionada

Beltramelli (2018) utiliza uma combinação de CNN e redes LSTM para geração de múltiplos *tokens* individuais a partir de uma UI (Figura 31).

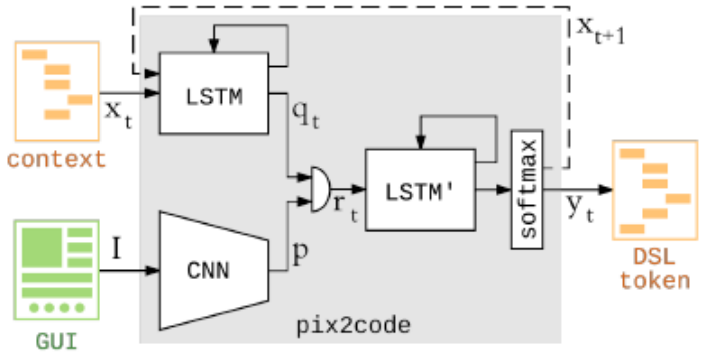


Figura 31 - Visão Geral do modelo pix2code (BELTRAMELLI, 2018)

Liu et al. (2018) se baseia fortemente na arquitetura de Beltramelli (2018), substituindo as redes LSTM por redes BLSTM visando otimização desta arquitetura (Figura 32).

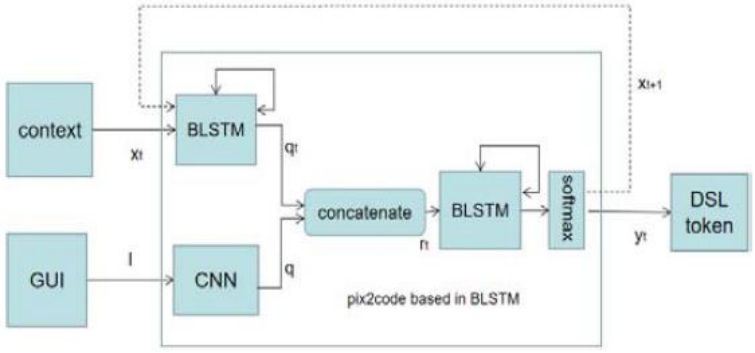


Figura 32 - Visão Geral do modelo pix2code BLSTM (LIU et al., 2018)

Moran et al. (2018) utiliza de uma combinação de técnicas de visão computacional clássicas para detecção de componentes e construção da hierarquia e uma CNN para classificação dos componentes individuais (Figura 33).

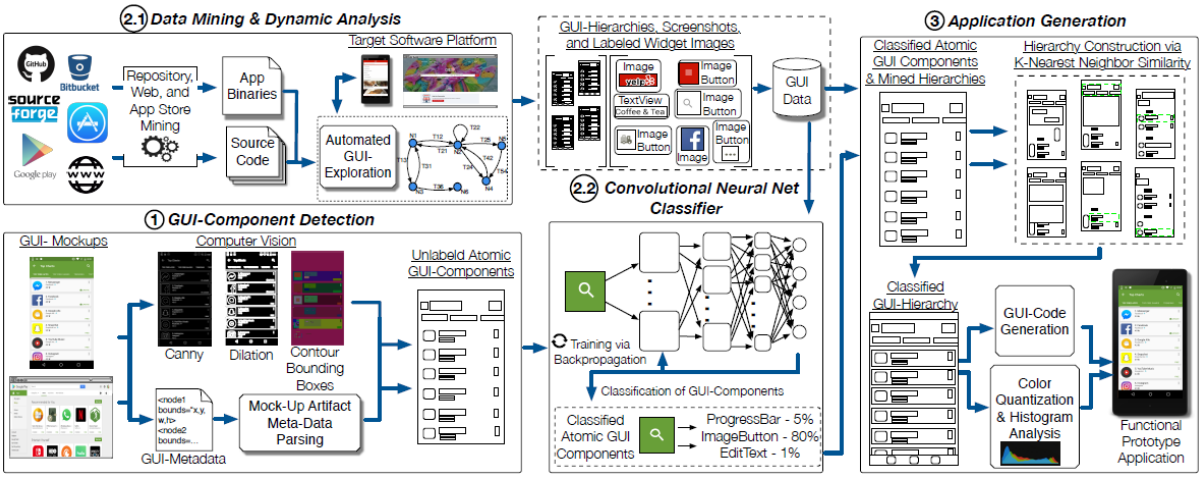


Figura 33 - Abordagem proposta para prototipação de GUIs automatizada (MORAN et al., 2018)

Yun et al. (2018) utiliza de um modelo de rede pré-treinada YOLO (EHSANI et al., 2018) para detecção dos componentes de interface, mas não entra em detalhes sobre o modelo utilizado para geração de código.

Robinson (2019) realiza testes com diversas arquiteturas e modelos de rede, optando pela utilização de uma rede Xception (CHOLLET, 2017) como *backbone* para uma arquitetura DeepLabv3+ (Chen et al., 2018) (Figura 34).

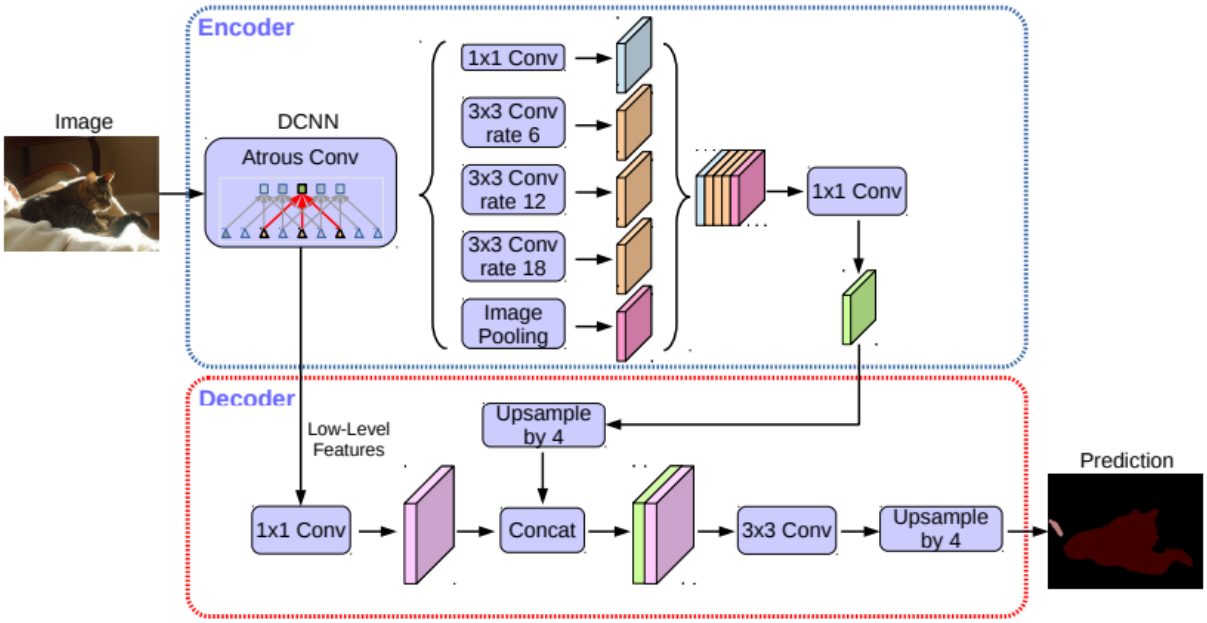


Figura 34 - Arquitetura de rede DeepLabv3+ (Chen et al., 2018)

HUANG et al. (2019), por outro lado, se baseia em um modelo de rede CNN profunda VGG-A, utilizando duas redes idênticas para comparação do *sketch* com *screenshots* da base de dados (Figura 35).

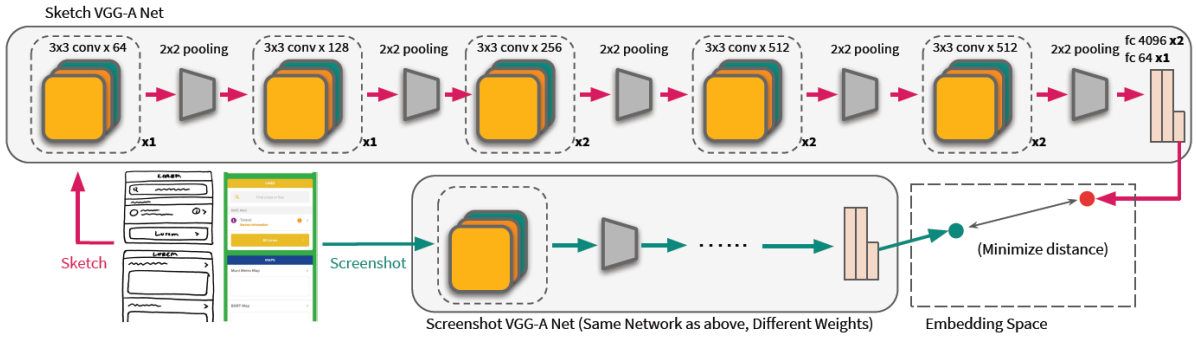


Figura 35 - Arquitetura de rede da NN Swire (HUANG et al., 2019)

Nguyen et al. (2015) utiliza exclusivamente técnicas de visão computacional clássica, portanto não apresenta um modelo de rede neural, enquanto Kim et al. (2018) não entra em muitos detalhes sobre a arquitetura de rede utilizada, dizendo apenas que se trata de uma *Faster R-CNN*. Similarmente, Ge (2019) diz apenas que foram utilizadas técnicas de *deep learning* para geração de esqueletos de interface.

Quanto aos tipos de aprendizagem, é possível observar que todos os artefatos que utilizam *machine learning* utilizam alguma forma de aprendizagem supervisionada, sendo a aprendizagem não supervisionada utilizada apenas em casos complementares, em sistemas compostos por múltiplas redes.

PA6. Como foi medida a qualidade do resultado e quais resultados foram obtidos?

Dentre os artefatos relevantes, se observou grande dispersão nas medidas de desempenho e técnicas de avaliação utilizadas (Tabela 15).

Tabela 15 - Dados extraídos para responder à Pergunta de Análise 6

Citação	Dados		
	Medidas de desempenho	**Avaliação do modelo**	**Resultados da avaliação**
(BELTRAMELLI, 2018)	NI	NI	NI
(GE, 2019)	Ranking de similaridade do APK utilizado como base	Seis APKs do Repositório escolhidos como base e desenhados a mão	APK base costuma aparecer nos 6 mais similares retornados
(KIM et al., 2018)	NI	NI	NI
(NGUYEN et al., 2015)	Runtime, Similaridade pixel-a-pixel de screenshots e similaridade da hierarquia da UI entre aplicativo original e o aplicativo gerado	488 Screenshots de Teste (Sendo 16 Sketches) dos Top 100 apps	UIs geradas são similares às originais tanto em comparação pixel-a-pixel quanto em termos de sua hierarquia
(HUANG et al., 2019)	Comparação com técnicas clássicas e avaliação de experts	Similaridade das interfaces encontradas com o sketch	SWIRE demonstra melhor performance que técnicas clássicas, e geralmente experts ficaram satisfeito com os melhores resultados encontrados (Os que o SWIRE acusa maior similaridade)
(ROBINSON, 2019)	3 medidas: 1. Quão bem cada método detecta e classifica elementos e containers, 2. Quanto a estrutura do site gerado corresponde à do site original, e 3. Quão bem a cada abordagem lida com exemplos não vistos.	250 Sketches gerados automaticamente + 16 desenhados a mão	Comparação entre as técnicas utilizadas
(LIU et al., 2018)	Função de avaliação do Keras	Função de avaliação do Keras	0.85
(YUN et al., 2018)	NI	NI	NI
(MORAN et al., 2018)	RQ1: Efetividade da CNN, RQ2: Construção hierárquica da GUI, RQ3: Similaridade visual, RQ4: Aplicação industrial	RQ1: Testes de precisão, RQ2: Comparação com outros modelos, RQ3: Comparação com outros modelos, RQ4: Opinião de especialistas	REDRAW é capaz de detectar e classificar componentes de UI, gerando aplicativos visualmente e hierarquicamente similares ao esperado, e demonstrou impactos positivos na indústria

Alguns artefatos sequer realizaram uma avaliação do modelo implementado, como Beltramelli (2018) e Kim et al. (2018).

Ge (2019) realiza uma avaliação simples, escolhendo seis *screenshots* da sua base de dados, fazendo *sketches* com base nos *screenshots* escolhidos e verificando em que posição o *screenshot* original ficou na lista de *screenshots* visualmente mais similares com o *sketch* retornado pelo sistema (Figura 36), concluindo que geralmente

o sistema obtém performance adequada, com todos os *screenshots* originais entre os 6 primeiros resultados.

UI scenario	Target App package name	Similarity score*	Rank*
login & register	com.choiceoflove.dating	81.08%	1
search	com.parfield.prayers.lite	64.86%	5
setting	com.google.android.apps.messaging	72.97%	5
list	com.baviux.pillreminder	67.57%	6
picture	com.kudago.android	78.42%	1
detail	com.google.android.play.games	70.27%	1

* Scores and ranks of the six target GUIs in the final search result list, respectively.

Figura 36 - Resultados da avaliação do sistema (GE, 2018)

Nguyen et al. (2015) realizou avaliação com 488 *screenshots* de teste, sendo 16 destes *sketches* desenhados a mão, retirados dos Top 100 aplicativos mais populares. Na avaliação foram observados o *runtime*, a similaridade pixel-a-pixel e a similaridade hierárquica das UIs geradas.

Moran et al. (2018) também realizou avaliação da similaridade pixel-a-pixel e hierárquica, mas avaliou também a efetividade da CNN utilizada para classificação de componentes, bem como a possibilidade de aplicação das técnicas na indústria, por meio da opinião de especialistas.

Liu et al. (2018) fornece como avaliação apenas o resultado da função de avaliação do *framework* utilizado, o que dificulta a comparação com outros métodos.

Huang et al. (2019) realiza uma avaliação diferente, pois assim como Ge (2019), este artefato não visa a geração de código, mas sim a pesquisa de interfaces similares. Assim, a avaliação é feita por meio da comparação com técnicas de visão computacional clássica. Para a avaliação, se realizou s*ketches* de 278 *sceenshots* da base de dados, e observou a proporção em que o sistema retornava o *screenshot* base no top 1 ou top 10 resultados (Figura 37). Ao final, se concluiu apenas que o

sistema possui um desempenho melhor que as técnicas de visão computacional utilizadas.

Technique	Top-1	Top-10
(Chance)	0.362%	3.62%
BoW-HOG filters	15.6%	38.8%
Swire	**15.9%**	**60.9%**

Figura 37 - Resultado da avaliação Swire (Huang et al., 2019)

Huang et al. (2019) realiza também uma avaliação por meio da opinião de especialistas da área, apresentando os resultados para *designers* em busca de comentários, em uma tentativa de observar os casos em que o sistema funciona bem e os casos em que este obtém pior desempenho.

Robinson (2019) apresenta diversas comparações parciais entre técnicas propostas, e em uma avaliação final foca na comparação entre as técnicas de visão computacional clássica e de *machine learning*. Dividindo sua avaliação em *micro performance* e *macro performance*. Na etapa de avaliação de *micro performance*, Robinson (2019) foca na detecção e classificação individual de elementos e *containers*, enquanto na etapa de avaliação de *macro performance* o foco é na similaridade do *website* gerado com aquele utilizado de base e na capacidade de extrapolação da técnica, quando confrontada com exemplos novos. Robinson (2019) concluiu que as técnicas de visão computacional clássica utilizadas obtiveram melhor performance desempenho na classificação de *containers*, enquanto as técnicas de *machine learning* se sobressaíram na classificação de elementos, bem como na similaridade dos *websites* gerados. As técnicas de visão computacional clássica utilizadas também demonstraram melhor capacidade de extrapolação, algo que talvez possa ser alterado com modificações no *dataset* utilizado.

PA7. Qual a forma de implantação da rede/modelo de *machine learning*?

Dentre os artefatos avaliados, poucos indicam um modelo de implantação do sistema no meio de produção (Tabela 16).

Tabela 16 - Dados extraídos para responder à Pergunta de Análise 7

	Dados	
Citação	**Descrição de como o modelo é implantado**	**Link**
(BELTRAMELLI, 2018)	Disponível como um sistema web (Uizard)	https://uizard.io/
(GE, 2019)	NI	
(KIM et al., 2018)	NI	
(NGUYEN et al., 2015)	Protótipo de programa disponível online.	http://pixeltoapp.com/
(HUANG et al., 2019)	NI	
(ROBINSON, 2019)	Disponível como um sistema web (Sketch2Code)	https://sketch2code.azurewebsites.net/
(LIU et al., 2018)	NI	
(YUN et al., 2018)	NI	
(MORAN et al., 2018)	Discussões com especialistas da área sobre possível utilização	

Beltramelli (2018), apesar de não tratar diretamente em seu artigo sobre o meio de implantação, está disponível o modelo desenvolvido como um sistema web por meio de assinatura, chamado Uizard. De maneira similar, Robinson (2019) disponibiliza também um sistema web chamado Sketch2Code.

Nguyen et al. (2015) não disponibiliza um sistema completo, mas um protótipo está disponível online para experimentação.

Os demais artefatos não apresentam um modelo de implantação, apenas Moran et al. (2018), apesar de não apresentar planos concretos para implantação, apresenta discussões com especialistas de algumas das principais empresas da área sobre a possível utilização de um sistema similar no meio de produção. Concluindo que ocorreria um possível aumento da produtividade.

3.4. Discussão

Os resultados dessa revisão sistemática demonstram que ainda são poucos e recentes os trabalhos que desenvolvem um modelo completo de geração automática de *wireframes* a partir de *sketches*. Mesmo também já existindo abordagens voltados a app moveis não foi encontrado nenhum modelo para a geração de *wireframes* no App Inventor.

Diversos trabalhos encontrados têm um foco parcialmente similar, mas objetivos diferentes do esperado. Ge (2019) e Huang et al. (2019), por exemplo, realizam a geração de um modelo intermediário a partir de *sketches*, mas apenas como um meio de obter os *screenshots* similares em sua base de dados, e não para geração de *wireframes*.

Os trabalhos encontrados demonstram grande variação nos modelos de *machine learning* utilizados, apontando o grande número de possíveis abordagens ao problema em questão. A única forte similaridade identificada é a utilização de algum modelo de CNN para processamento de imagens.

Muitos dos trabalhos selecionados focam no processo de desenvolvimento da solução, e não em detalhes do modelo resultante, resultando em falta de detalhes sobre o pré-processamento dos dados, estrutura da rede neural e geração de códigos em muitos destes.

De forma geral, os trabalhos encontrados pecaram na avaliação e divulgação dos resultados, sem boas métricas ou com número de amostras insuficiente, dificultando a comparação entre os modelos encontrados.

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, através da utilização de diversos sinônimos de conceitos relevantes e clausulas lógicas. 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.
- **Pesquisador único:** O fato de que somente um pesquisador, o autor deste trabalho, participou desta revisão sistemática, pode comprometer a isenção da aplicação dos critérios de inclusão e exclusão.

4. MODELO DE GERAÇÃO AUTOMÁTICA DE WIREFRAMES NO APP INVENTOR A PARTIR DE SKETCHES

Esse capítulo apresenta a proposta o modelo desenvolvido para a geração automática de *wireframes* no App Inventor a partir de *sketches*.

4.1. Análise de Requisitos

O objetivo deste trabalho é o desenvolvimento de um modelo de geração automática de *wireframes* em App Inventor a partir de *Sketches*. Sendo este modelo dividido em duas etapas (Figura 38).

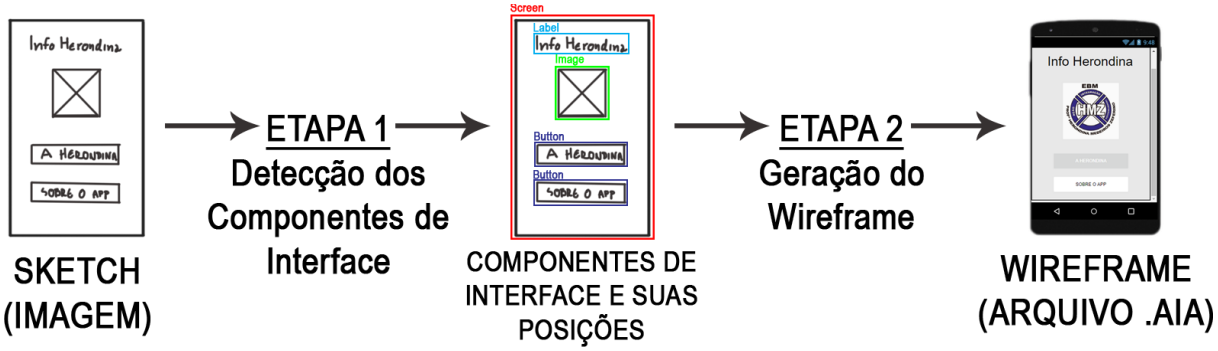


Figura 38 - *Workflow* alto nível do modelo desenvolvido

A primeira etapa tem como entrada um *sketch* de interface. Nesta etapa é realizada a detecção dos diferentes elementos de interface e suas posições no espaço. Definimos esta etapa como um problema de aprendizado de máquina.

Dentro do processo de prototipação esses sketches são desenhados à mão pelos usuários, sendo posteriormente fotografados, sendo estas fotografias fornecidas como entrada ao modelo. Este processo é realizado para cada tela do aplicativo, com a maioria dos aplicativos de App Inventor não ultrapassando 6 telas diferentes.

Segundo Mitchell (1997), um programa de computador aprende com uma experiência *E*, em relação a alguma classe de tarefas *T* e uma medida de performance

P, se sua performance em tarefas de *T*, medida por *P*, melhora de acordo com sua experiência *E*.

Nesta etapa, a Tarefa (T) é identificar as posições de diferentes componentes de interface em *sketches*. A Experiência (E) é um conjunto de *sketches* de interfaces de aplicativos em forma de fotografias dos sketches desenhados a mão com marcações das posições e tipos de seus componentes, e o desempenho (P) do modelo é medido pela similaridade dos componentes identificados com o valor verdade esperado (Os componentes originais e suas posições).

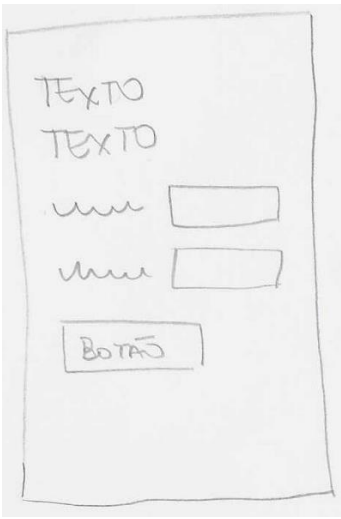
A segunda etapa trata da geração de *wireframes* em App Inventor. Nesta etapa, a Tarefa (T) é gerar automaticamente gerar o código .aia de um wireframe a partir das posições de diferentes componentes de interface detectados nos *sketches*. A Experiência (E) é um conjunto de arquivos .aia retirados da galeria App Inventor e da Iniciativa Computação na Escola, e o desempenho (P) é medido pela similaridade do arquivo .aia gerado com o resultado esperado a partir dos *sketches.* Esta etapa tem como entrada os componentes identificados na etapa anterior, e como saída um arquivo de projeto do App Inventor (.aia).

4.2. Etapa 1 - Detecção dos Componentes de Interface

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

4.2.1. Conjunto de dados

Para o treinamento do modelo é necessário um conjunto de dados apresentando duplas de sketches (Fotos) e um arquivo identificando o conjunto de *labels* com as posições dos diferentes componentes na imagem (Figura 39).




```
{
  "version": "4.2.10",
  "flags": {},
  "shapes": [
    {
      "label": "Label",
      "points": [
        [
          135.42857142857133,
          398.57142857142856
        ],
        [
          458.2857142857142,
          502.85714285714283
        ]
      ],
      "group_id": null,
      "shape_type": "rectangle",
      "flags": {}
    },
    {
      "label": "Label",
…
```


Figura 39 - Exemplo de Foto de Sketch e arquivo com conjunto de *labels* correspondente

Devido ao grande número de tipos de componentes de interface de usuário disponibilizados pelo App Inventor, e a falta de um padrão de *sketch* para grande parte destes, foram selecionados os componentes que aparecem com maior frequência em aplicativos de App Inventor, com base em (ALVES et al., 2020) (Figura 40).

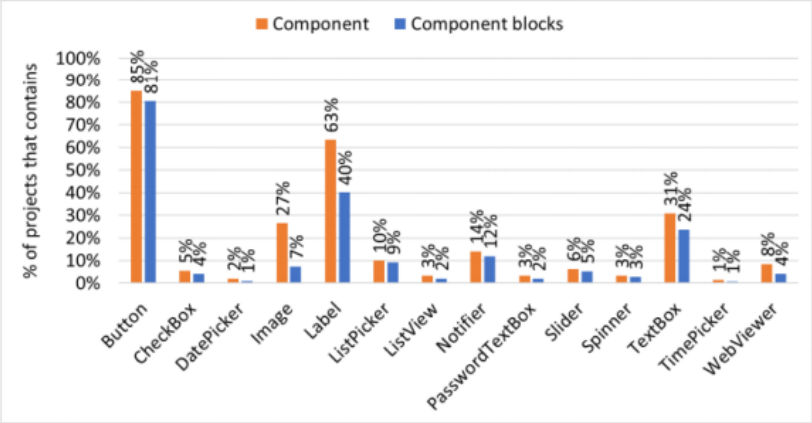
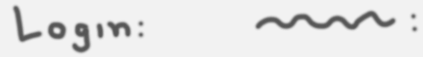


Figura 40 - Uso dos componentes de Interface de Usuário no App Inventor (ALVES et al., 2020)

Foram selecionados apenas os componentes utilizados em ao menos 5% dos projetos de App Inventor, com exceção do componente *Notifier,* que não foi considerado por não representar um elemento interno da interface de usuário, mas sim diálogos de alerta ou mensagens que sobrepõem a tela.

Adicionalmente, foram considerados os seguintes componentes, adicionados ao App Inventor após a realização da análise de (ALVES et al., 2020), e que aparentam ser amplamente utilizados em aplicativos recentes da MIT App Inventor *Gallery*: *Switch e Map*. Resultando nos componentes de interface de usuário a serem detectados pelo modelo listados na Tabela 17:

Tabela 17 - Componentes selecionados e Sketches correspondentes

COMPONENTE	**EXEMPLO DE SKETCHES**
Label	
Button	
Image	
Textbox	
ListPicker	
Switch	
CheckBox	
Slider	
Map	

Para a criação de um conjunto de dados adequado, foram obtidos de forma semiautomática *screenshots* de aplicativos criados pelo Grupo de Qualidade de Software (GQS) da Universidade Federal de Santa Catarina (UFSC) e de aplicativos no MIT App Inventor *Gallery*, com apenas os componentes selecionados (Figura 41).

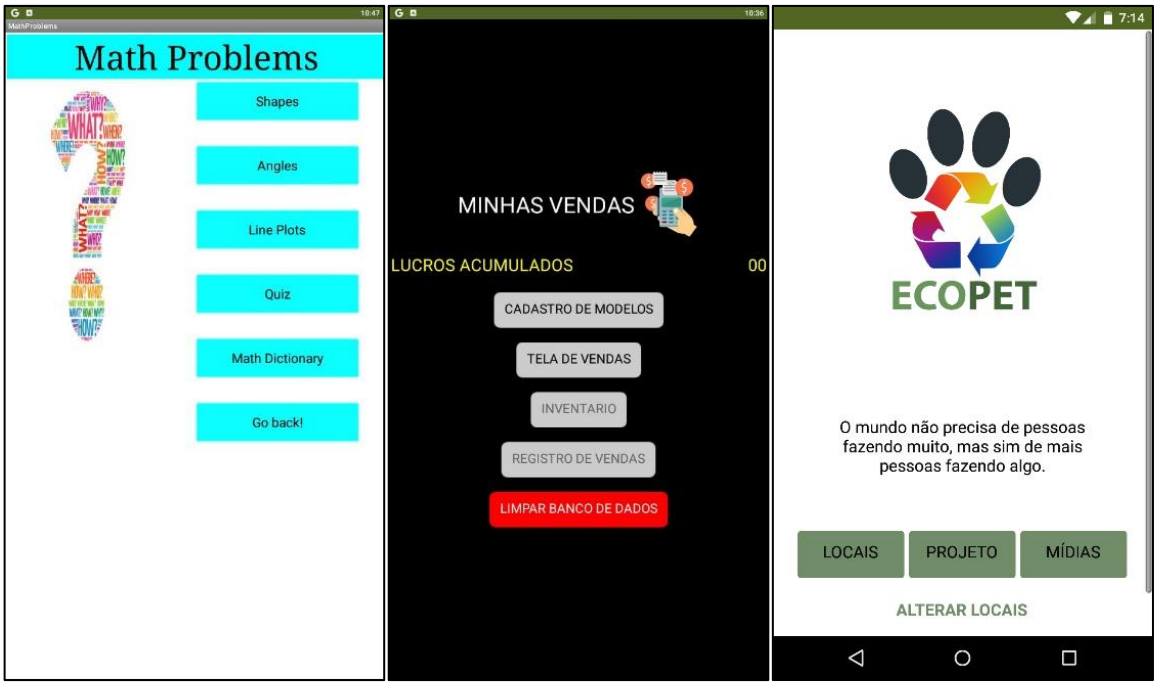


Figura 41 - Exemplos de *Screenshots* selecionados

Com base nestes *screenshots*, foram criados manualmente *sketches* das interfaces de usuários em papel e caneta por 29 voluntários (jovens e professores da educação básica) e membros do GQS (alunos e pesquisadores na área de computação e design). Cada um recebeu 5 capturas de telas selecionadas aleatoriamente e um manual com instruções e exemplos de sketches, sendo solicitado a estes a realização do desenho dos sketches referentes aos *screenshots* atribuídos. Por fim, os participantes fotografaram os sketches com seus celulares.

Esse processo resultou em um *dataset* com fotos de *sketches* realistas e em condições similares as esperadas na utilização do sistema final (Figura 42). Com uma boa variedade tanto no desenho dos *sketches* quanto na realização das fotografias (condições de iluminação e câmeras variadas).

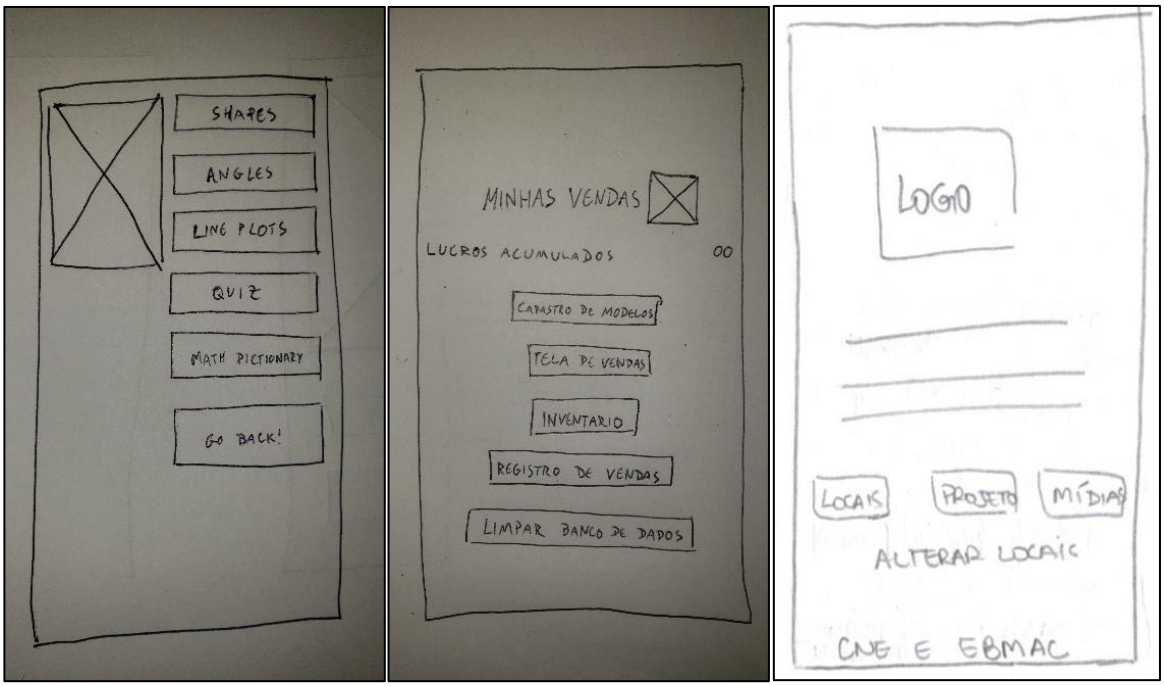


Figura 42 - Exemplos de Sketches coletados

Foram avaliados ainda os *datasets* disponibilizados por BELTRAMELLI (2018), HUANG et al. (2019), LIU et al. (2018) e MORAN et al. (2018) para complementar o conjunto de dados próprio, devido à presença tanto de interfaces e *sketches* reais quanto gerados de maneira automática, mas devido a grande diferença entre as interfaces presentes nestes *datasets* com as observadas em aplicativos de App Inventor, se descartou sua utilização.

No total, foram coletados 284 *sketches* de interface de aplicativos de App Inventor.

***Labeling* das imagens**

Para cada um dos *sketches* coletados, foi realizado manualmente o *labeling* dos diferentes componentes de interface presentes (Figura 43). O processo de *labeling* manual foi realizado utilizando o programa labelme (WADA, 2016), devido sua flexibilidade e facilidade de uso quando comparado a programas alternativos, resultando em um arquivo JSON para cada imagem, com informações sobre o

tamanho, posição e tipo de cada *label*, além de dados adicionais utilizados pelo programa. Devido a utilização do formato de arquivo JSON, estes arquivos podem ser facilmente acessados e convertidos para os formatos de entrada necessários para diferentes modelos de rede (Apêndice A).

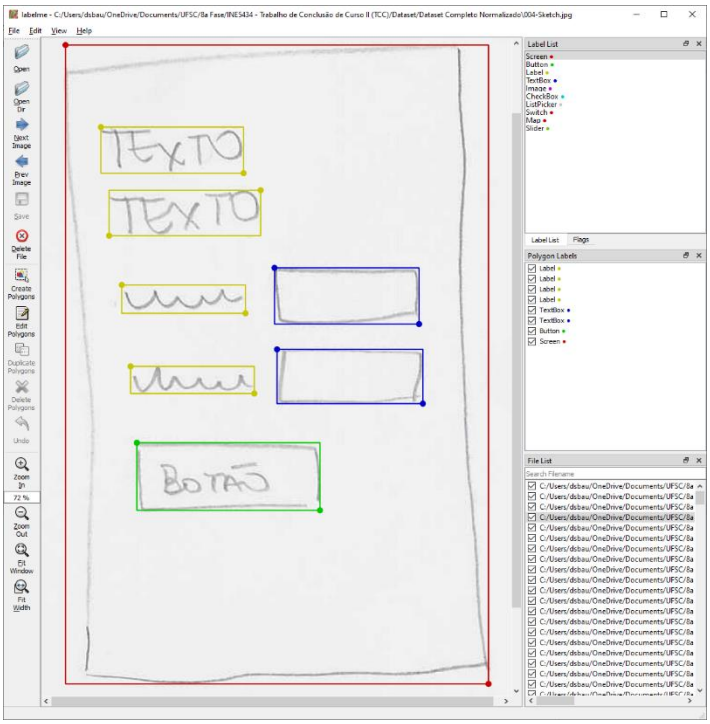


Figura 43 - *Labeling* dos *sketches*

Devido a utilização de *screenshots* como base para as interfaces do *dataset* e voluntários sem especializações na área de *design* de interfaces, diferenças entre os aplicativos originais e os *sketches* eram esperadas, principalmente em componentes parecidos, como botões e caixas de texto. Assim, apenas após a realização do processo de *labeling* foi possível realizar uma análise mais profunda da frequência de aparição de cada tipo de componente no dataset (Tabela 18).

Tabela 18 - Frequência de componente de interface no conjunto de dados

Componente	Número de ocorrências	Frequência
Label	687	34%
Button	708	35%
TextBox	178	9%
CheckBox	135	7%

Image	167	8%
Switch	36	2%
Slider	28	1%
Map	16	1%
ListPicker	40	2%
Total	1995	100%

Cada um dos 284 *sketches* presentes no conjunto de dados também contém uma *label "screen"*, delimitando os limites da tela do aplicativo. Estas *labels* não foram consideradas na análise de frequência por não representar um componente de interface do App Inventor, mas sim a tela do celular como um todo.

É possível observar que há uma grande diferença na frequência dos diferentes tipos de componentes de interface, com Botões (*Button*) e Legendas (*Labels)* compondo 69% dos componentes do conjunto de dados. A análise da frequência de utilização dos diferentes componentes de interface (Figura 40) mostra que esta diferença não caracteriza necessariamente um problema com o conjunto de dados, mas provavelmente se deve à diferença de frequência de utilização destes componentes em projetos do App Inventor, e consequentemente sua frequência de aparição nos *screenshots* utilizados como base para a criação dos *sketches*.

4.2.2. Modelo de Aprendizagem

Foi utilizado o *framework open-source* para redes neurais *Darknet*, escrito em C e Cuda, com otimizações para computação tanto em CPU e GPU (REDMON, 2016), para implantação de um modelo YOLO (*You Only Look Once*) de detecção de objetos em imagem. Isso permitiu o treinamento e utilização da rede sem necessidade de implantação de detalhes de mais baixo nível, por este trabalho se tratar de uma pesquisa na utilização de técnicas de *deep learning* no processo de *design* de interfaces de usuário, e não uma pesquisa em *deep learning*.

Seleção do Modelo

Existem diversos modelos YOLO, desenvolvidos para diferentes aplicações. Tiny-YOLO, por exemplo, utiliza uma rede neural mais simples visando menor tempo de detecção e peso computacional, ideal para aplicações de vídeo em tempo real. Como o sistema será utilizado apenas para imagens individuais, e tempo de detecção pequeno não é a sua maior prioridade, foi utilizada o modelo YoloV3 (REDMON et al., 2019), que utiliza uma rede *Darknet-53* (Figura 44).

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 44 - Arquitetura da rede Darknet-53 (REDMON et al., 2019)

Durante o desenvolvimento posterior do sistema foi publicado o modelo YoloV4, mas nos testes realizados com o mesmo conjunto de dados, a performance obtida

com este modelo mais recente foi um pouco inferior, logo, foi selecionado o modelo YoloV3 como arquitetura de rede para o sistema final.

Transferência de aprendizado

Como o conjunto de dados utilizado é relativamente pequeno, com apenas 284 imagens, há um risco considerável de *overfitting* se realizado o treinamento apenas com as imagens do conjunto. Para solucionar esse problema, exploramos a transferência de aprendizado a partir de uma rede YOLO pré-treinada com o conjunto de dados ImageNet (REDMON et al. 2019), um conjunto de dados com mais de 1 milhão de imagens com objetos manualmente rotulados em 1000 categorias (RUSSAKOVSKY et al., 2014). Esta estratégia permite utilizar um conjunto de dados pequenos para treinar uma rede efetivamente (YOSINSKI et al. 2014).

Treinamento

O *fine-tuning* da rede foi realizado de acordo com as orientações de BOCHKOVSKIY (2019), com alteração das camadas convolucionais e YOLO para se adequar ao número de classes de objetos considerados (9 tipos de componentes de UI, além da tela como um todo). As camadas YOLO foram configuradas para 10 classes, enquanto as camadas convolucionais imediatamente anteriores a elas foram configuradas com 45 filtros, seguindo a fórmula de BOCHKOVSKIY (2019):

$$\text{filters} = (\text{classes} + 5) \times 3 = (10 + 5) \times 3 = 45$$

Seguindo BOCHKOVSKIY (2019), o treinamento foi subdividido em batches de 64 imagens, com 16 subdivisões para contornar limitações de memória. O treinamento foi realizado até 20.000 batches, seguindo a fórmula de BOCHKOVSKIY (2019):

$$\text{max batches} = \text{classes} \times 2000 = 10 \times 2000 = 20000$$

O conjunto de dados foi dividido aleatoriamente em um conjunto de treinamento com 237 imagens (~ 85%) e um conjunto de validação com 42 imagens (~15%) (Apêndice B).

O treinamento da rede foi realizado no Google Colab, com os valores dos pesos da rede salvos a cada 1000 batches, para posterior avaliação da evolução de seu desempenho.

Avaliação de Desempenho

São utilizadas diversas métricas diferentes para medir a performance de um modelo de rede, com a medida de qualidade de um modelo de detecção de objeto sendo tipicamente baseada no IoU (*Intersection Over Union*), uma medida baseada no índice Jaccard que avalia a sobreposição entre duas caixas delimitadoras, a caixa delimitadora verdade e a caixa delimitadora prevista (ZOU et al. 2019). Outra métrica frequentemente usada para detecção de objetos é a Precisão Média *(Average Precision - AP*) (LIU et al. 2020). O AP é definido como a precisão de detecção média em diferentes recalls e geralmente é avaliado de uma maneira específica da classe de objeto. Para comparar as classes gerais de objetos de desempenho, o AP médio (mAP) calculado sobre todas as classes de objetos é comumente utilizado como a métrica final de desempenho.

Além do IoU e mAP, o F1 score também é uma métrica utilizada para a avaliação deste tipo de modelo. O F1 score é a média harmônica da precisão e recall, transmitindo informações sobre o equilíbrio entre estes (ROBINSON 2018). Um F1 score alto indica que o sistema executa uma detecção que é exata e completa (ROBINSON 2018).

Durante a criação do modelo foram analisadas todas essas métricas permitindo também a comparação com diversos modelos existentes, apesar de estes variarem

na métrica utilizada. A Figura 4 mostra a evolução do desempenho do modelo Sketch2aia durante o treinamento de acordo com estas métricas.

Figura 45 - Evolução do desempenho durante o treinamento

O melhor resultado obtido foi após 18.000 batches, apresentando o melhor F1 score de 0.88 com valores mais balanceados de IoU médio e mAP, quando comparado aos demais resultados (Figura 5).

Resultados após 18000 Batches	
F1-Score	0.88
Average IoU	75.85%
mAP	87.72%
AP Screen	97.62%
AP Label	80.88%
AP Button	97.10%
AP TextBox	81.44%
AP CheckBox	95.21%
AP Image	100.00%
AP Switch	100.00%
AP Slider	100.00%
AP Map	100.00%
AP ListPicker	25.00%

Figura 46 - Melhores resultados obtidos (total e por componente)

A Figura 5 mostra a precisão média obtida para cada componente no conjunto de teste. A presença de componentes com 100% de precisão provavelmente se deve

ao tamanho limitado do conjunto de teste e consequentemente baixa frequência desses componentes (p.ex. somente 4 ListPickers), mas ainda indica um bom resultado. Outro possível fator para a baixa precisão obtida com alguns componentes é a ausência de características distintas quando comparados aos demais. ListPicker, Button e TextBox, por exemplo, são muito similares visualmente com somente pequenos detalhes determinando a diferença entre eles.

Comparando com outros modelos existentes, o F1 score médio obtido por nosso modelo (0.88) é maior que o melhor índice obtido por Robinson (2018) (0.81, em imagens) e que o índice médio obtido por Huang et al. (2019) (0.775). Quanto ao único modelo encontrado que também usou a métrica mAP (SULERI et al. 2019) (PANDIAN, SULERI 2020), que obteve uma mAP de 84.9%, enquanto o modelo Sketch2aia obteve uma mAP de 87.72%. Dessa forma, o modelo Sketch2aia demonstra resultados de avaliação de desempenho equivalente ou melhor que os modelos existentes encontrados na revisão da literatura, mostrando um suporte a detecção de componentes de UI com precisão.

4.3. Etapa 2 – Geração de Código de *Wireframes*

A saída da detecção de componentes de UI em *sketches* é uma representação intermediária, na forma de uma lista de elementos detectados, seus respectivos níveis de confiança e suas posições na imagem (coordenadas do ponto central, largura e altura) conforme ilustrado na Figura 6.

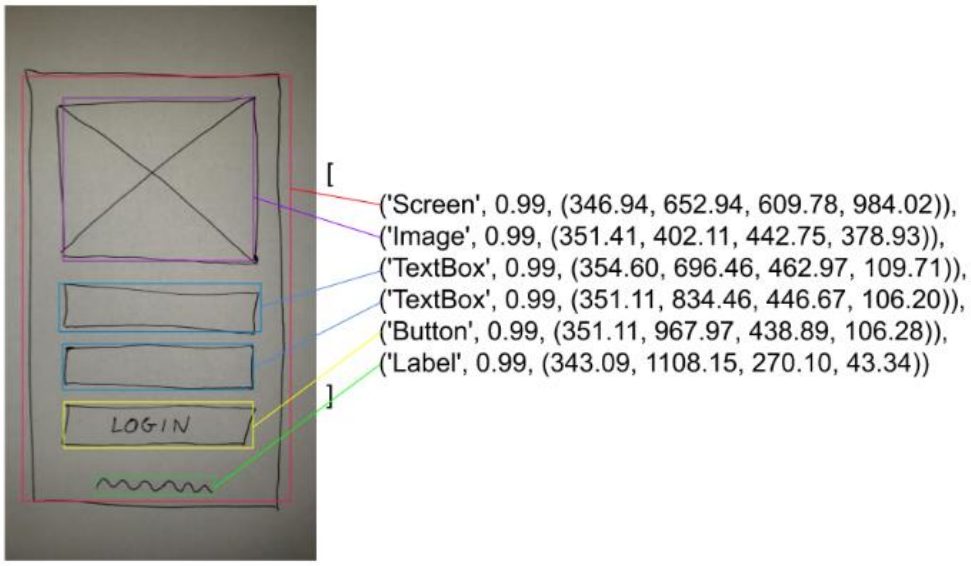


Figura 47 - Exemplo de saída da detecção de componentes de UI em *sketches*

O primeiro passo para gerar o código de *wireframes* App Inventor é a eliminação de possíveis sobreposições de componentes, para isso checamos para cada par de elementos se a área de sobreposição deles é maior que 50% da área do menor elemento entre eles, eliminando o com menor índice de confiança em caso afirmativo.

Mesmo após a eliminação das sobreposições, os componentes não estão organizados em forma de *layouts*, mas por suas posições no espaço explicitamente definidas. Esta representação não é compatível com o App Inventor, que requer que os componentes estejam organizados em um dos seguintes *layouts* para obter uma organização mais complexa:

- OrganizaçãoHorizontal;
- *HorizontalScrollArrangement*;
- OrganizaçãoEmTabela;
- OrganizaçãoVertical;
- *VerticalScrollArrangement*.

Devido a sua simplicidade e comportamento mais previsível, utilizamos nessa implementação somente a OrganizaçãoHorizontal e a OrganizaçãoVertical. Para organizar os componentes em *layouts* verticais e horizontais, é utilizado um algoritmo recursivo (Figura 7).

Com os componentes organizados em *layouts* compatíveis, é realizada para cada *sketch* uma tradução direta para um arquivo .scm, que define o *wireframe* da tela correspondente dentro do arquivo .aia (Figura 8). O arquivo .scm do App Inventor encapsula uma estrutura JSON que contém todos os componentes visuais utilizados no aplicativo (MUSTAFARAJ et al. 2017), preenchido com valores determinados empiricamente, por meio da análise de diversos projetos de App Inventor (Figura 8). Para cada tela é gerado também um arquivo .bky vazio. O arquivo .bky encapsula uma estrutura XML com todos os blocos de programação usados na lógica do aplicativo (MUSTAFARAJ et al. 2017).

```
função alinhar(componentes, orientação)
inicio
        // Criar uma lista para representar os componentes do alinhamento atual
        alinhamentoAtual = lista()
        // Ordenar os componentes da lista de acordo com a orientação do alinhamento atual
        componentes = ordenarLista(componentes, orientação)
        // Enquanto a lista de componentes não estiver vazia
        para cada componenteAtual em componentes
        inicio
                componentes.remove(componenteAtual)
                // Verifica se existem na lista componentes alinhados com o atual no sentido contrário
                componentesAAnalisar ← lista()
                novosComponentes ← lista()
                componentesAAnalisar.adiciona(componenteAtual)
                // Para cada componente alinhado novo, verifique também se existem outros componentes alinhados com este
                para cada componenteAnalisado em componentesAAnalisar
                inicio
                        componentesAAnalisar.remove(componenteAnalisado)
                        para cada componente em componentes
                        Inicio
                                // Se os componente estiver alinhado e não tiver sido verificado, adicione e verifique
                                se componenteAnalisado.estaAlinhado(componente, orientação) então
                                        se componente não está em novosComponentes então
                                                componentesAAnalisar.adiciona(componente)
                                                novosComponentes.adiciona(componente)
                                        fim_se
                                fim_se
                        fim_para
                Fim_para
                // Se não estiver alinhado com outros componentes, adicione na lista, senão alinhe estes primeiro
                se novosComponentes.vazia() então
                        alinhamentoAtual.adiciona(componenteAtual)
                senão
                        componentes ← componentes - novosComponentes
                        novosComponentes.adiciona(componenteAtual)
                        alinhamentoAtual.adiciona(alinhar(novosComponentes, ~orientação))
                fim_se
        fim_para
        retorna tupla(orientação, alinhamentoAtual)
fim
```

Figura 48 - Algoritmo para agrupamento dos elementos detectados em layouts compatíveis com o App Inventor

Após a geração dos códigos de *wireframe* correspondentes a todas as telas, o arquivo de propriedades é gerado conforme as configurações do projeto (Nome, Tela Principal etc.).


```
{
    "authURL": [
        "ai2.appinventor.mit.edu"
    ],
    "YaVersion": "206",
    "Source": "Form",
    "Properties": {
        "$Name": "Screen1",
        "$Type": "Form",
        "$Version": "27",
        "AppName": "Test",
        "Title": "Screen1",
        "Uuid": "0",
        "$Components": [
            {
                "$Name": "OrganizacaoVertical1",
                "$Type": "VerticalArrangement",
                "$Version": "3",
                "AlignHorizontal": "3",
                "AlignVertical": "1",
                "Height": "-2",
                "Width": "-2",
                "Uuid": "6204202716",
                "$Components": [
                    {
                        "$Name": "ListaSuspensa1",
                        "$Type": "Spinner",
                        "$Version": "1",
                        "AlignHorizontal": "1",
                        "AlignVertical": "1",
                        "Height": "-1",
                        "Width": "-1",
                        "Uuid": "7956601802"
                    },
```


Figura 49 - Exemplo de código de *wireframe* para App Inventor

Os arquivos gerados são compactados em um arquivo .zip seguindo a estrutura de arquivos esperada pelo App Inventor, sendo este salvo como um arquivo .aia que pode ser importado diretamente no App Inventor.

4.4. Ferramenta Web

Como resultado, a ferramenta Sketch2aia foi implementada como uma aplicação web. A ferramenta permite que o usuário faça o *upload* de até 6 imagens de *sketches* de UI de aplicativos móveis. O Sketch2aia então detecta automaticamente os componentes de UI e gera o código de App Inventor (um arquivo .aia)

correspondente. O arquivo pode ser então importado pelo usuário no ambiente App Inventor, onde ele(a) pode continuar o desenvolvimento do aplicativo (Figura 50).

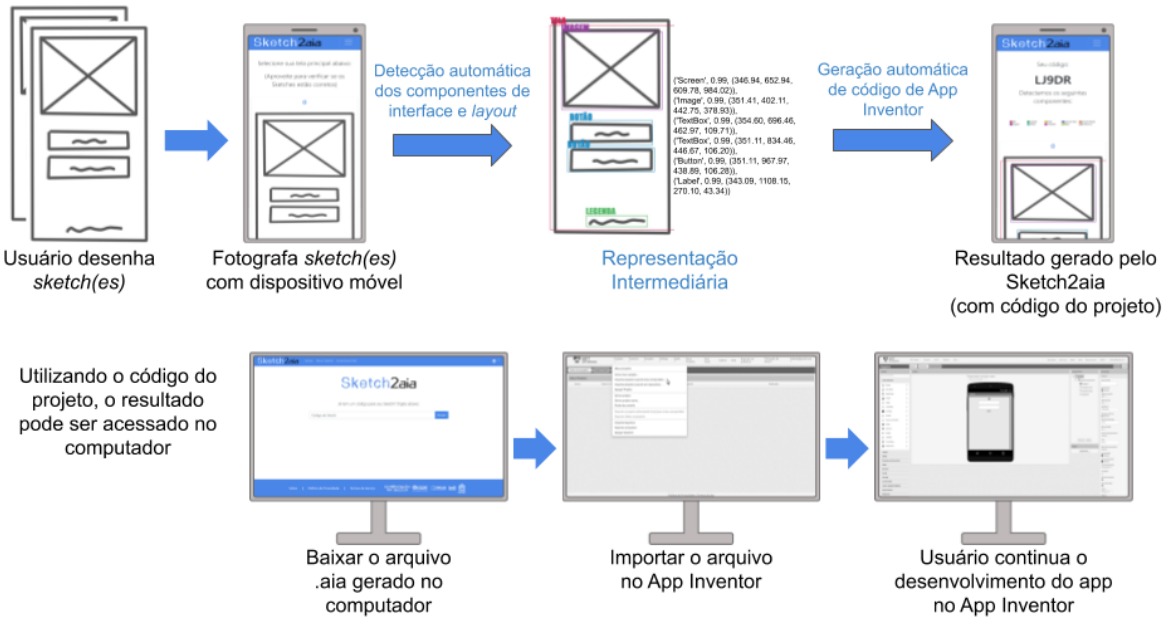


Figura 50 - Workflow do Sketch2aia

Ao acessar a ferramenta, o usuário é informado deste workflow esperado em sua utilização, por meio da página mostrada na Figura 51.

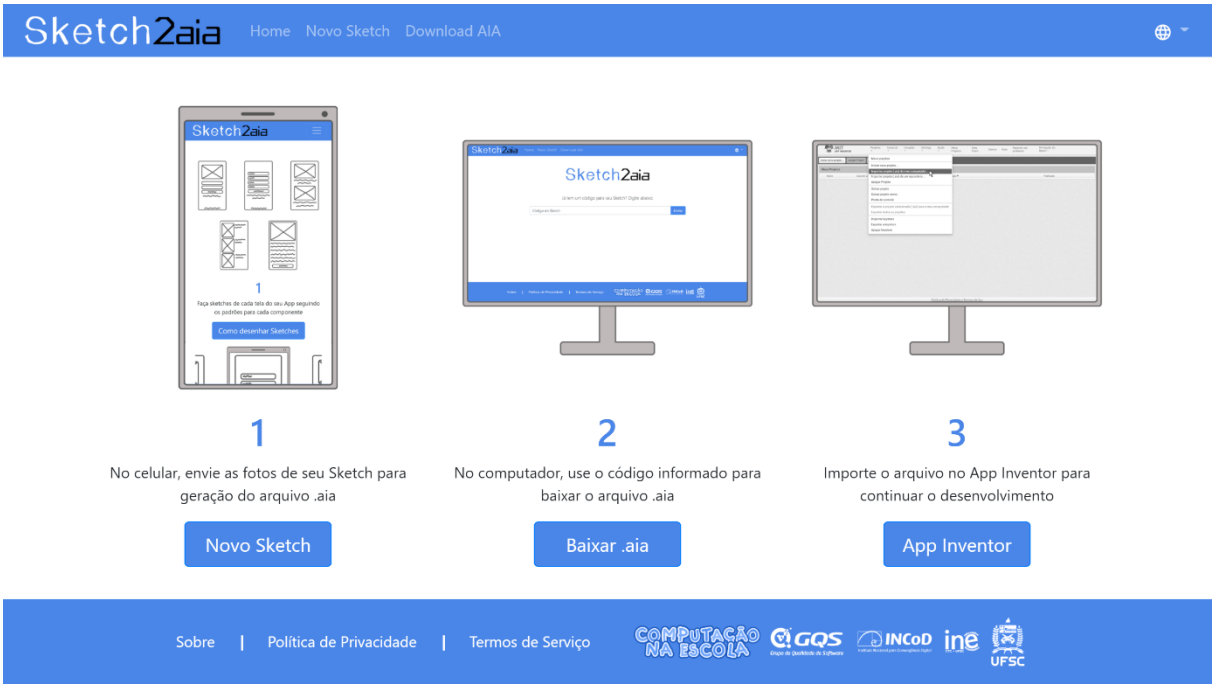


Figura 51 - Página inicial da ferramenta web Sketch2aia

A utilização esperada da ferramenta é dividida em 3 etapas, sendo a primeira realizada preferencialmente no aparelho celular do usuário, e as demais no computador. A primeira etapa trata do envio dos *sketches* para geração do código de App Inventor, e é sugerido sua realização no celular para eliminar a necessidade de envio das fotos para o computador, uma vez que estas são geralmente fotografadas com o celular. Caso sejam utilizados *sketches* digitais ou as fotos já se encontrem no computador, não é necessária a utilização do celular, podendo o processo inteiro ser concluído no computador.

Ao selecionar a opção “Novo *Sketch*”, o usuário é direcionado para a página de criação de um novo *sketch* mostrada na Figura 52.

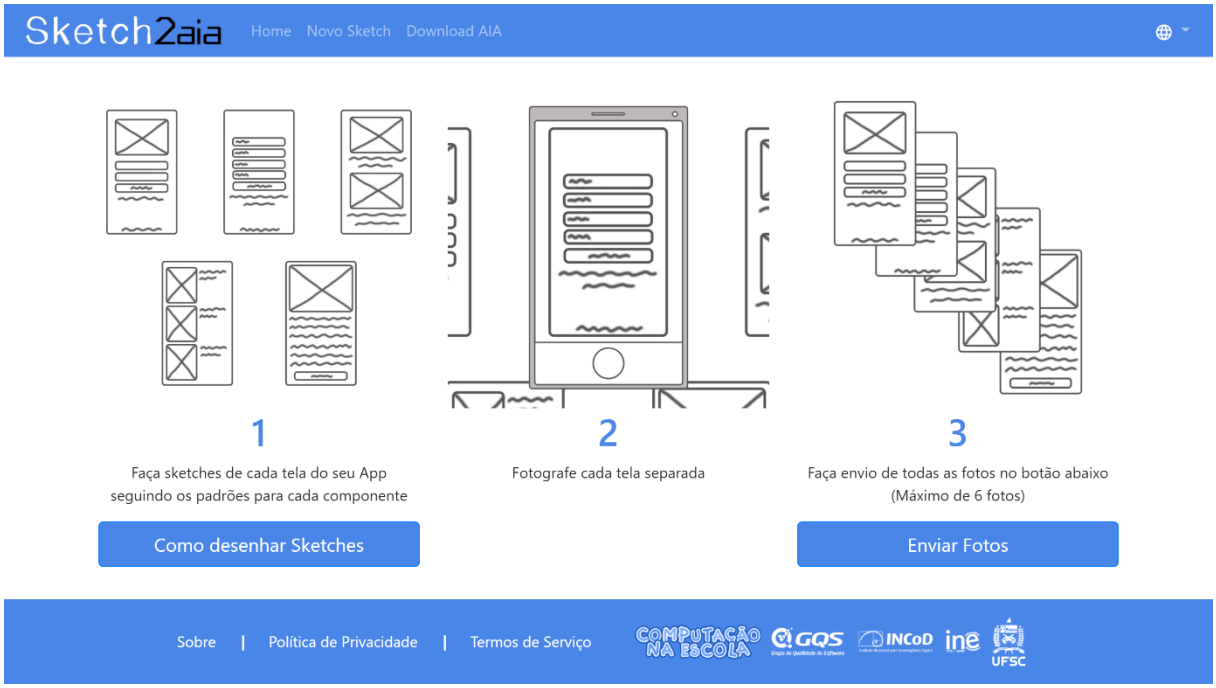


Figura 52 – Página de novo sketch da ferramenta Sketch2aia

Esta página contém informações sobre o processo de desenho e fotografia dos *sketches*, com a possibilidade de acessar instruções mais detalhadas por meio do botão “Como desenhar Sketches” (Figura 53).

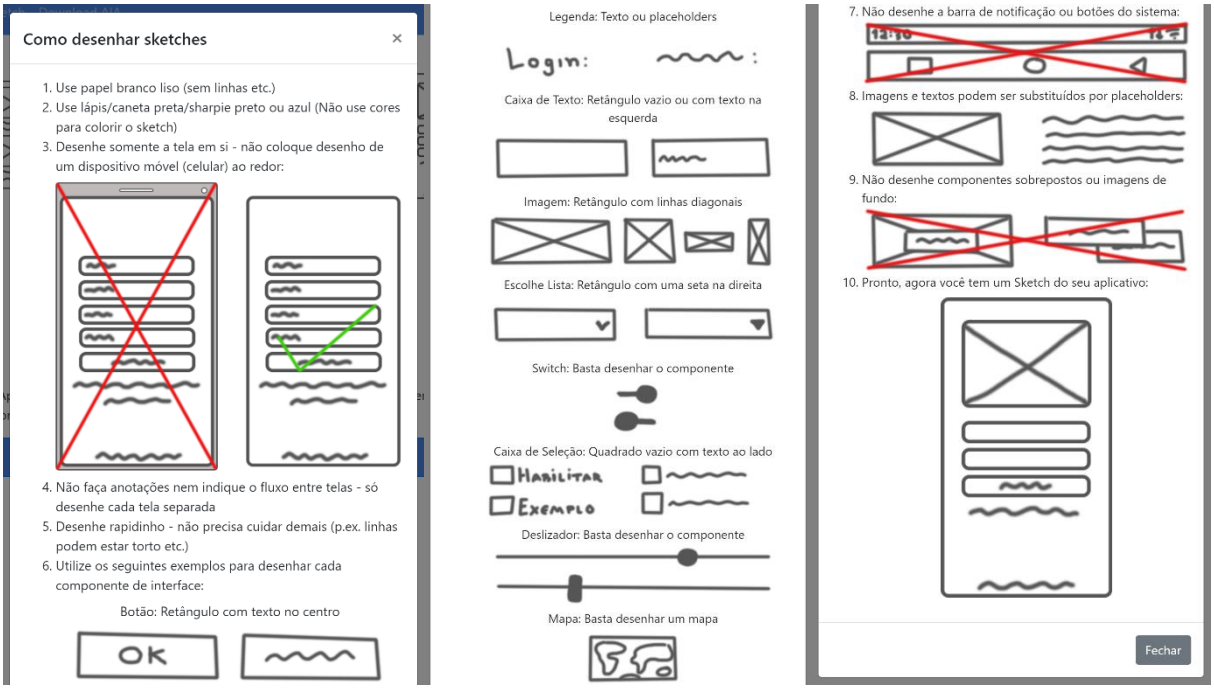


Figura 53 – Janela com instruções de como desenhar *Sketches*

Uma vez que o usuário desenhou e fotografou todos os *sketches*, deve realizar o envio destes por meio do botão “Enviar Fotos”, sendo redirecionado para a página de configuração do projeto mostrada na Figura 54.

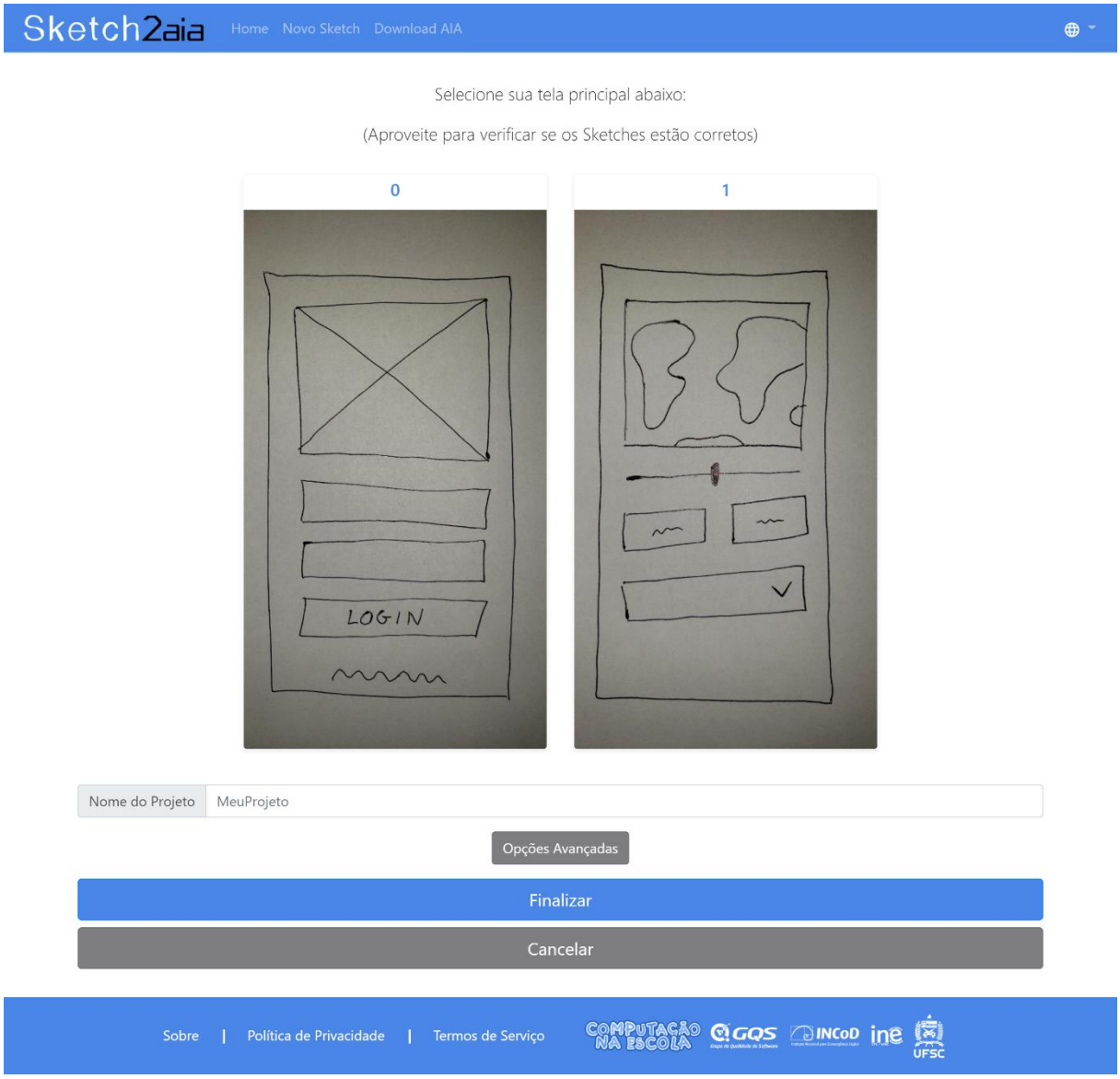


Figura 54 - Página de configuração do projeto da ferramenta Sketch2aia

Nesta página, a ferramenta permite que o usuário verifique seus sketches e configure algumas opções, como o nome do projeto, a tela principal, e a escolha entre a utilização de EscolheLista ou ListaSuspensa, dois componentes com função semelhante no App Inventor, mas esteticamente diferentes. Estas duas últimas opções, devido a necessidade de conhecimentos mais detalhados sobre App Inventor são acessadas por meio do botão “Opções Avançadas” (Figura 55).

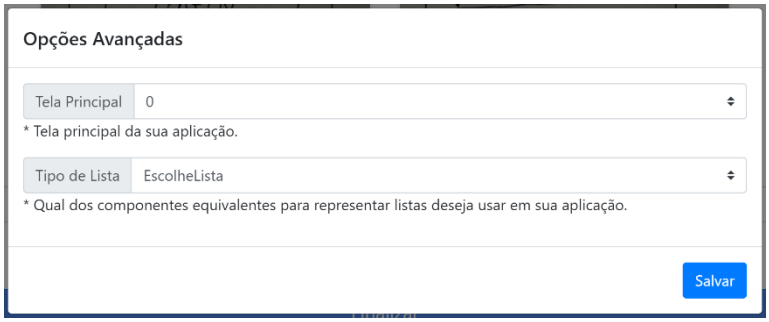


Figura 55 - Opções avançadas da ferramenta Sketch2aia

Concluídas as configurações do projeto, o usuário pressiona o botão “Finalizar”, e a geração do código de App Inventor correspondente é realizada. Finalizada a detecção, o usuário é redirecionado para a página de download do arquivo .aia. Caso esta primeira etapa tenha sido realizada no celular, o usuário pode utilizar o código informado nesta página (Figura 56) para acessá-la diretamente no computado.

Figura 56 - Página de download do arquivo aia com código para acesso direto

Além disso, é possível visualizar uma *preview* da detecção realizada, possibilitando a identificação de diferenças entre os componentes esperados e os

realmente detectados antes da sua importação para o App Inventor. O código gerado pela ferramenta também possibilita o fácil compartilhamento de projetos. Para acessar *sketches* gerados previamente, basta pressionar o botão “Download AIA” na barra de navegação a qualquer momento, ou o botão “Baixar aia” na tela inicial (Figura 51).

Figura 57 - Tela para acesso à arquivos .aia gerados previamente

Ao digitar o código informado na tela do celular na barra de pesquisa e pressionar o botão “Enviar”, o usuário é redirecionado para a página de seu projeto que está em seu celular, possibilitando o download do arquivo .aia diretamente no computador, facilitando assim o processo de importação no App Inventor.

Assim como no celular, o usuário pode novamente verificar a correspondência da detecção com os componentes esperados, para em seguida realizar o download do arquivo (Figura 58).

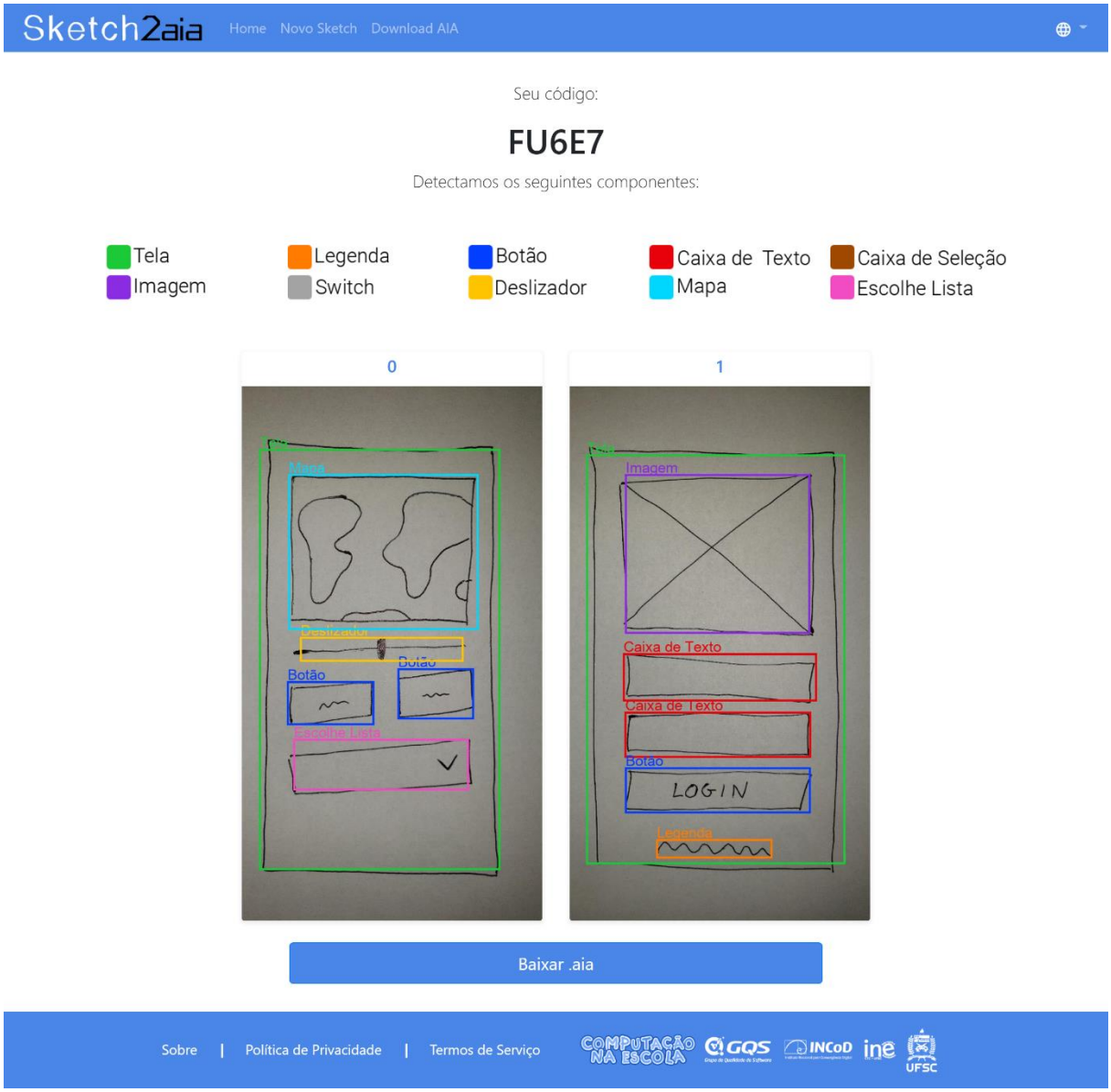


Figura 58 - Resultado fornecido pela ferramenta Sketch2aia

Com o arquivo .aia baixado em seu computador, o usuário pode então acessar o App Inventor e realizar a importação de seu projeto, dando continuidade ao desenvolvimento de seu App, como mostra a Figura 59.

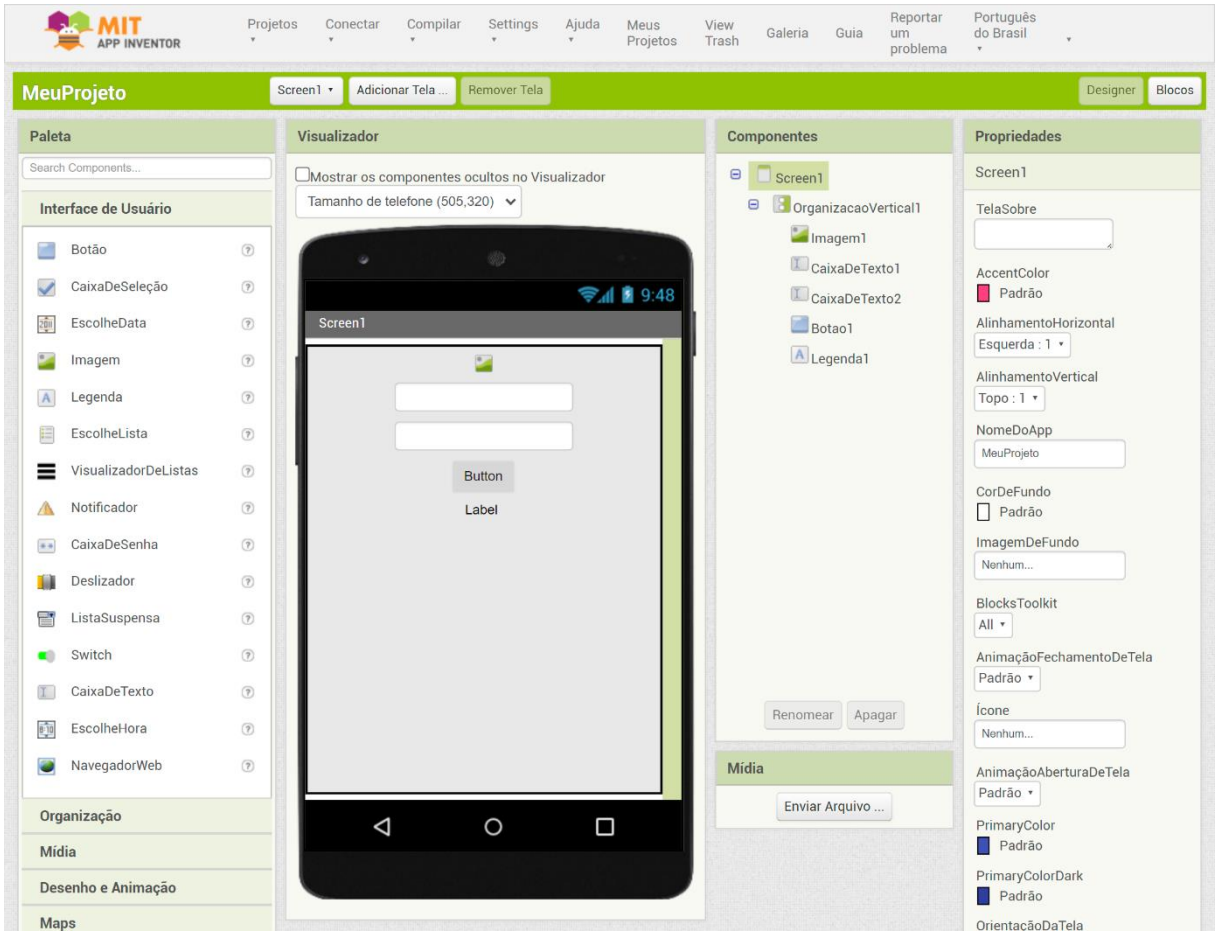


Figura 59 - Arquivo .aia gerado pelo Sketch2aia importado no App Inventor (MIT, 2020)

A ferramenta *Sketch2aia* é dividida em apenas dois grupos, por se tratar de uma ferramenta própria sem necessidade de integração a outros sistemas. Seus componentes são:

- Container Web: Contém o módulo de apresentação, módulo de detecção e o framework de redes neurais Darknet.
- *Browser*: Consiste no navegador de internet do usuário, onde a interface gráfica é exibida, e onde são feitos os downloads e uploads de artefatos pelo usuário.

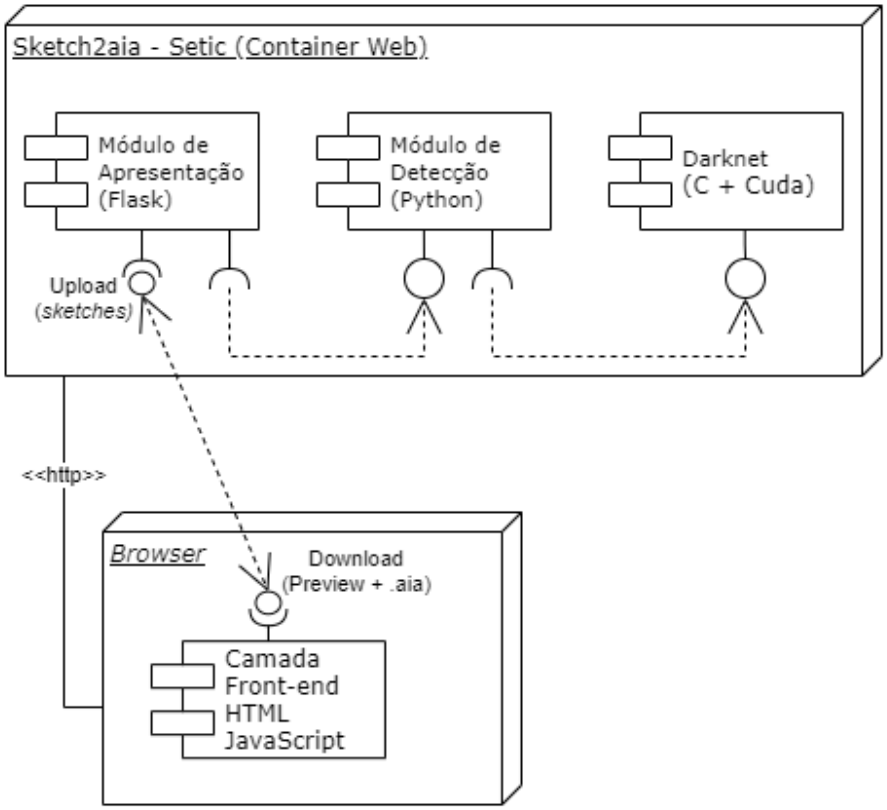


Figura 60 - Diagrama de Módulos do Sketch2aia

O módulo de apresentação da ferramenta web foi implementada em Python, utilizando o framework web Flask, devido a facilidade de uso e flexibilidade de invocação de demais módulos Python em sua execução.

A camada Front-end tem como função o desenvolvimento das páginas web apresentadas no navegador de internet do usuário. Esse módulo utiliza HTML5, CSS3, Bootstrap e Java Script.

O módulo de detecção é responsável pela invocação da rede *Darknet* que por sua vez analisa e detecta os componentes de interface em *sketches*, retornando suas posições na imagem. Com a saída da rede, o módulo de detecção é também responsável por gerar os layouts e o código de App Inventor correspondentes. Este módulo foi programado em Python, e utiliza o *wrapper* Python disponibilizado pelo *Darknet*, com algumas modificações, para a invocação da rede.

A rede *Darknet* é executada externamente ao código Python, sendo invocada para a realização da detecção em imagens.

A ferramenta está disponível online: http://www.gqs.ufsc.br/sketch2aia/.

5. AVALIAÇÃO DO SKETCH2AIA

5.1. Definição da Avaliação

A avaliação do sistema foi realizada com o objetivo de avaliar os seguintes fatores de qualidade:

- OA1. Correspondência entre o resultado do Sketch2aia (Wireframe .aia) e *sketch*;
- OA2. Tempo de processamento da ferramenta;
- OA3. Usabilidade da ferramenta;
- OA4. Vantagens e desvantagens da utilização da ferramenta no processo de desenvolvimento de um app.

O objetivo da realização de uma avaliação preliminar consiste em avaliar se os *wireframes* gerados correspondem ao design da UI do *sketch*. Esta avaliação foi feita por meio de um estudo de usuário nesta avaliação preliminar, uma vez que a correspondência entre interfaces se trata de uma questão subjetiva e difícil de ser avaliada por métricas objetivas. Além de uma avaliação subjetiva pelos usuários, o sistema foi analisado também por meio de testes de performance.

Para a realização deste estudo de usuário, são apresentados aos participantes 10 pares de *sketches* e capturas de tela dos *wireframes* gerados com Sketch2aia. Esses foram selecionados aleatoriamente a partir do conjunto de teste/validação. Os participantes do estudo também são solicitados a desenhar um *sketch* de um aplicativo móvel e, em seguida, usar a ferramenta web Sketch2aia com uma foto deste *sketch* gerando o *wireframe.* Na sequência, solicita-se que eles baixem o código .aia criado e o importem no ambiente do App Inventor, executando a interface do usuário do aplicativo em seus *smartphones*.

Em paralelo foram coletados dados para a avaliação do grupo de participantes selecionados para o estudo, sendo estes dados demográficos sobre a escolaridade dos usuários, sua área de conhecimento e sua familiaridade com o App Inventor.

Referente ao objetivo de avaliação OA1, na avaliação dos 10 pares de *sketch* e capturas de telas e na geração de *wireframe* próprio, os participantes classificaram a correspondência de cada um dos *wireframes* de UI com os *sketches* em uma escala ordinal de 4 pontos, variando de não corresponde a corresponde totalmente.

Dados referentes ao tempo de processamento da ferramenta (OA2) foram obtidos por meio de perguntas objetivas, bem como por meio de testes de performance realizados externamente a análise de usuário.

Enquanto a avaliação da usabilidade (OA3) do sistema foi realizada utilizando o questionário padronizado *System Usability Score* – SUS (REF) composto por 10 afirmações:

1. Eu acho que gostaria de usar esse sistema com frequência;
2. Eu acho o sistema desnecessariamente complexo;
3. Eu achei o sistema fácil de usar;
4. Eu acho que precisaria de ajuda de uma pessoa com conhecimentos técnicos para usar o sistema;
5. Eu acho que as várias funções do sistema estão muito bem integradas;
6. Eu acho que o sistema apresenta muita inconsistência;
7. Eu imagino que as pessoas aprenderão como usar esse sistema rapidamente;
8. Eu achei o sistema atrapalhado de usar;
9. Eu me senti confiante ao usar o sistema;

10. Eu precisei aprender várias coisas novas antes de conseguir usar o sistema.

Sendo cada pergunta respondida em uma escala Likert de 5 pontos, variando de Discordo Totalmente a Concordo Totalmente.

Avaliações adicionais quanto aos pontos fortes e fracos da ferramenta (OA4), além de possíveis benefícios e desvantagens de sua utilização durante o processo de desenvolvimento de um app foram coletadas por meio de comentários descritivos dos participantes do estudo.

5.2. Execução da Avaliação

O estudo com usuários ocorreu no contexto da iniciativa Computação na Escola/INCoD/INE/UFSC em Florianópolis/SC em junho de 2020. A população do estudo foi composta por 29 alunos e professores com formação em ciência da computação e/ou design, bem como 6 jovens e 1 professor representando usuários-alvo de um contexto da educação básica. O estudo foi realizado on-line, fornecendo instruções detalhadas aos usuários, junto com um questionário (Apêndice C). A participação foi voluntária.

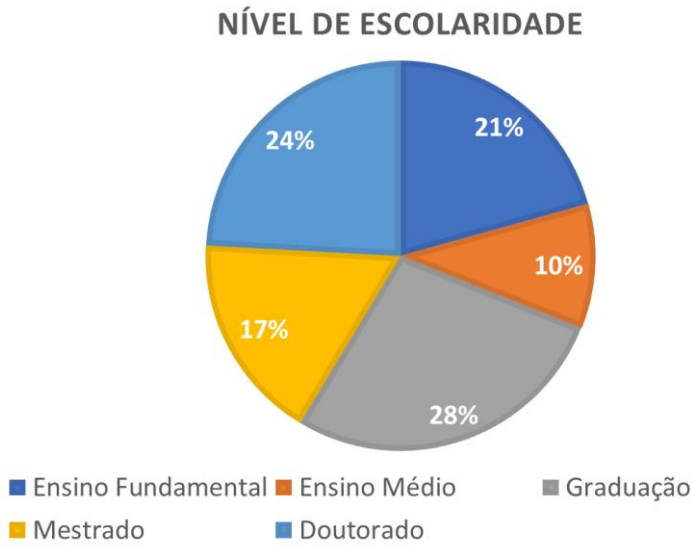


Figura 61 - Nível de Escolaridade dos participantes do estudo

É possível observar que os participantes do estudo tinham níveis de escolaridade variados, com membros de todos os níveis de estudo, desde alunos do fundamental até doutores (Figura 61).

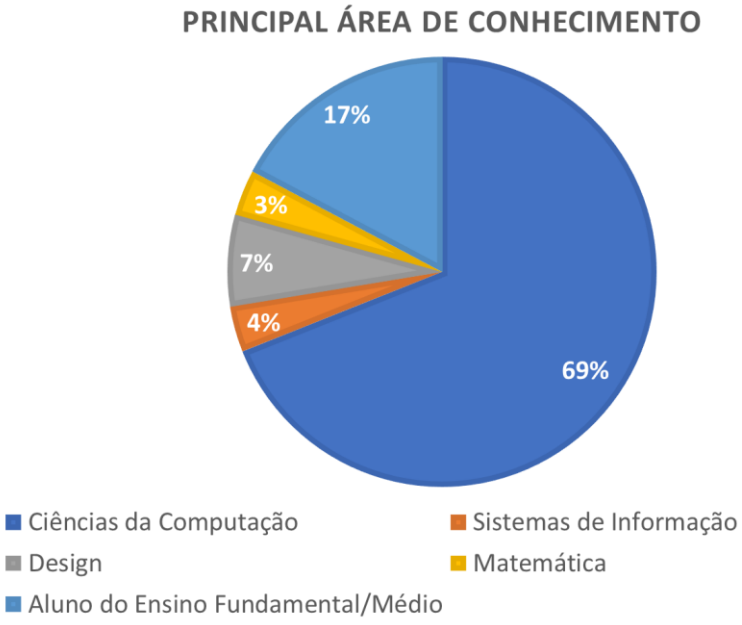


Figura 62 - Principal Área de Conhecimento dos participantes do estudo

Quanto à área de conhecimento, os participantes do estudo são compostos principalmente por pessoas da área de Ciências da Computação/Sistemas da Informação, mas o estudo também contou com a participação de pessoas de outras

aulas, como Design e Matemática (Figura 62). As respostas também mostram a presença de Alunos do Ensino Fundamental e Médio que não consideraram se enquadrar em nenhuma área de conhecimento específica.

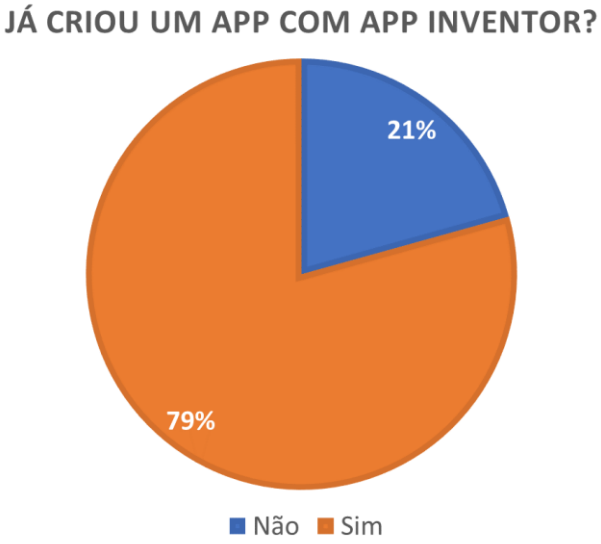


Figura 63 - Experiência dos participantes do estudo com o App Inventor

É possível observar também que a grande maioria dos participantes tinha alguma experiência com o App Inventor, por meio da criação de ao menos 1 aplicativo.

5.3. Resultados

5.3.1. Correspondência entre o resultado do Sketch2aia (*Wireframe* .aia) e *sketch*

Avaliando a similaridade entre 10 pares de *sketches* e wireframes gerados, com base nas respostas dos participantes, podemos observar que de forma geral os *wireframes* gerados correspondem largamente ou totalmente aos 10 *sketches* nos 10 exemplos fornecidos. A avaliação de correspondência com *sketches* próprios, mostrou resultados similares sendo observados com os *sketches* desenhados pelos próprios participantes, como mostra a Figura 64.

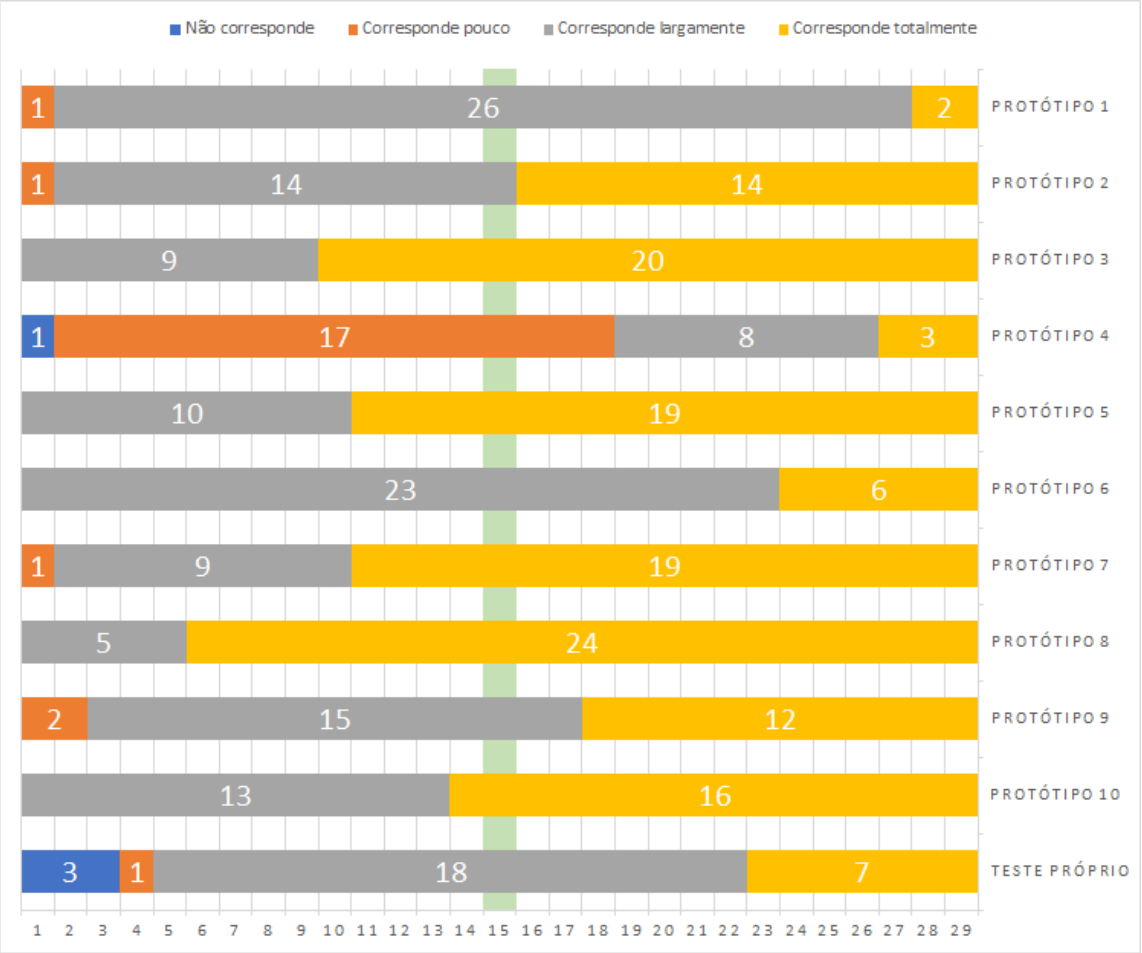


Figura 64 - Resultados da avaliação preliminar da correspondência entre *sketches* e *wireframes* de App Inventor gerados pela ferramenta

Uma exceção é o protótipo 4, que corresponde pouco segundo a maioria dos voluntários (Figura 65). Esta avaliação provavelmente se deve à detecção incorreta de *CheckBox* na parte superior da imagem, devido ao estilo de desenho utilizado para *Label* no *sketch*, com a combinação de letras e *placeholders.*

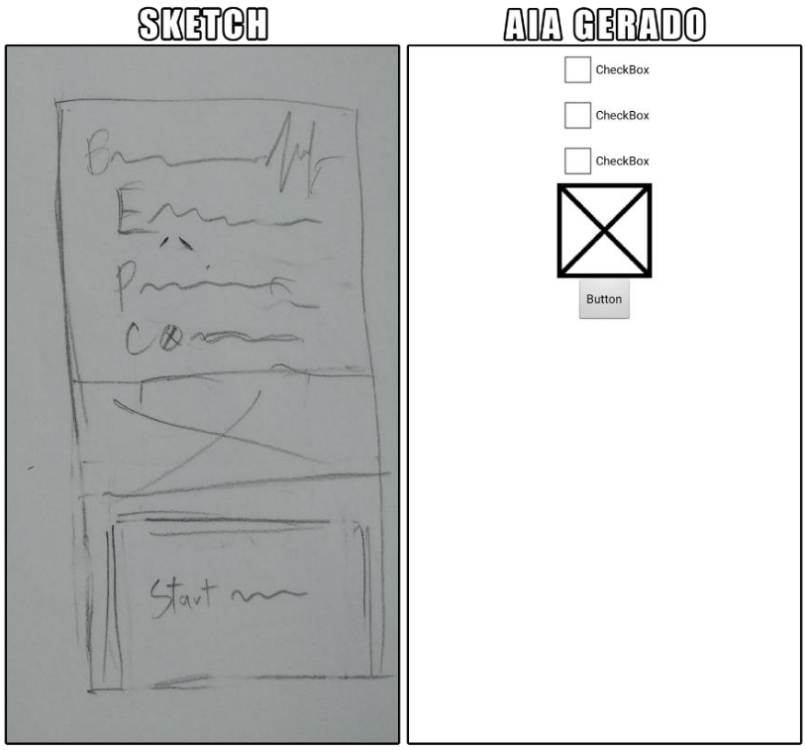


Figura 65 - *Sketch* e *wireframe* gerado do Protótipo 4

A Figura 66, por outro lado, mostra o protótipo 8, que segundo a avaliação dos participantes é o protótipo que mais corresponde ao *sketch* utilizado.

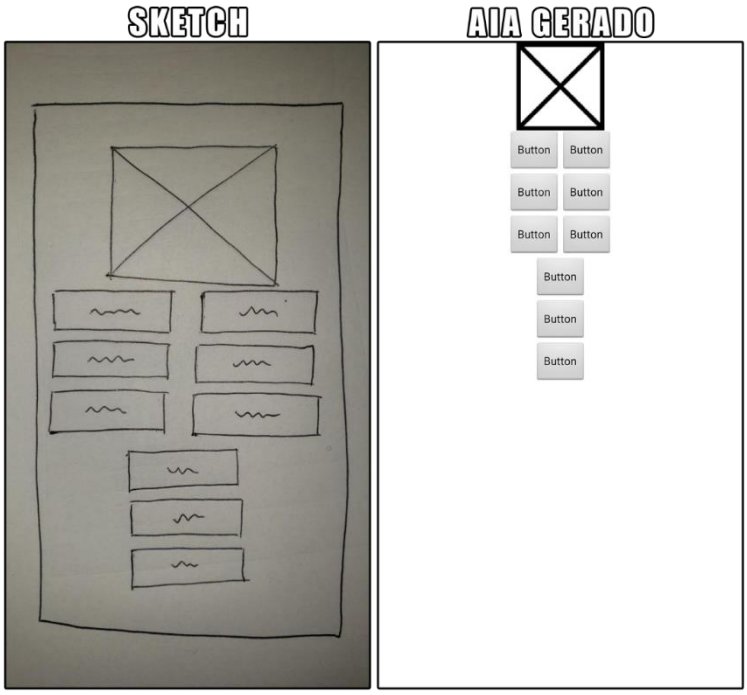


Figura 66 - *Sketch* e *wireframe* gerado do Protótipo 8

Neste protótipo, a detecção de componentes teve 100% de precisão, com as únicas diferenças entre o *sketch* e o *wireframe* ocorrendo na posição dos componentes.

No geral, a maior parte dos problemas identificados no estudo de usuário se referem ao *layout* e posicionamento dos componentes, e não a sua identificação. Estes problemas podem ser resolvidos por meio da inclusão de espaços em branco para melhor aproximação do *sketch*.

5.3.2. Tempo de processamento da ferramenta

Os participantes também avaliaram a performance do sistema como satisfatória, com apenas 6,9% dos participantes discordando. Em nossos testes o sistema leva em média 57.57 segundos para gerar o wireframe de um aplicativo com 3 telas, como mostra a Tabela 19.

Tabela 19 - Tempo para gerar o arquivo .aia para um aplicativo de 3 telas no Sketch2aia

	Tempo de Processamento (s)
Tentativa 1	57.32
Tentativa 2	56.57
Tentativa 3	57.16
Tentativa 4	57.17
Tentativa 5	58.77
Tentativa 6	56.79
Tentativa 7	57.34
Tentativa 8	56.63
Tentativa 9	57.1
Tentativa 10	59.87
Média	**57.47**

5.3.3. Usabilidade da ferramenta

Também foram obtidos resultados positivos quanto a avaliações do processo de uso da ferramenta. 100% dos participantes avaliaram que a ferramenta Sketch2aia

é útil no processo de desenvolvimento de apps e 96.6% avaliaram que a ferramenta é fácil de usar.

Para avaliar a usabilidade da ferramenta, foi adotado o SUS (*System Usability Scale*), que obteve uma pontuação média de 92,16 (Tabela 20). O SUS é uma ferramenta efetiva e confiável para medir a usabilidade de uma grande variedade de produtos e serviços (BANGOR et al., 2009), com a ferramenta Sketch2aia tendo sua usabilidade avaliada como excelente, após a conversão da pontuação SUS para uma escala adjetiva, como mostra a Figura 67.

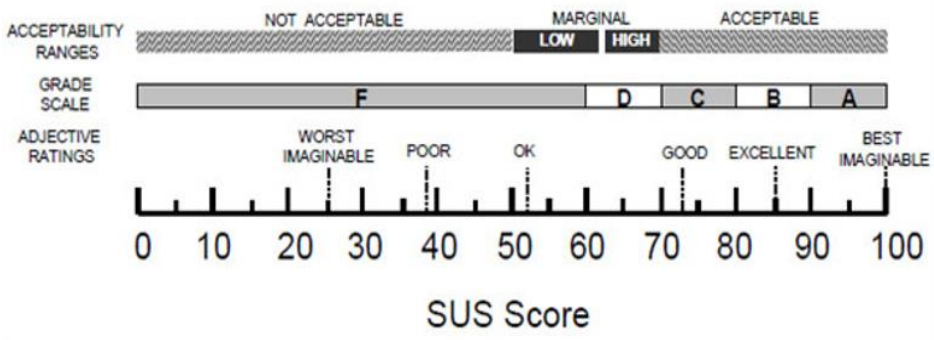


Figura 67 - Relação entre as avaliações adjetivas, pontuações de aceitabilidade, notas de escola e a pontuação SUS média (BANGOR et al., 2009)

A Tabela 20 mostra as pontuações SUS individuais fornecidas por cada participante do estudo de usuário.

Tabela 20 - Cálculo do *System Usability Score* (SUS)

	Pontuação total		**Pontuação total**
Participante 1	97,50%	Participante 16	95,00%
Participante 2	100,00%	Participante 17	82,50%
Participante 3	100,00%	Participante 18	85,00%
Participante 4	92,50%	Participante 19	92,50%
Participante 5	95,00%	Participante 20	92,50%
Participante 6	82,50%	Participante 21	90,00%
Participante 7	100,00%	Participante 22	97,50%
Participante 8	100,00%	Participante 23	85,00%
Participante 9	87,50%	Participante 24	87,50%
Participante 10	92,50%	Participante 25	85,00%
Participante 11	100,00%	Participante 26	92,50%
Participante 12	90,00%	Participante 27	100,00%
Participante 13	72,50%	Participante 28	92,50%

Participante 14	97,50%	Participante 29	87,50%
Participante 15	100,00%	**Média**	**92,16%**

5.3.4. Pontos fortes e oportunidades de melhoria da ferramenta

Durante o estudo de avaliação foram coletados também os pontos fortes e fracos da ferramenta, além de possíveis benefícios e desvantagens de sua utilização durante o processo de desenvolvimento de um app foram coletadas por meio de comentários descritivos dos participantes. A Tabela 21 mostra as justificativas dos participantes para sua avaliação anterior da ferramenta.

Tabela 21 - Justificativas para a avaliação da ferramenta

Explique sua resposta:
A ferramenta é muito útil para automatizar a tarefa chata de passar o sketch para um wireframe no App Inventor.
A ferramenta torna mais rápido e fácil um passo importante de criação da base da UI para o aluno
Acredito que principalmente na área em que estou estudando, o wireframe feito de forma mais rápida facilitaria muito o processo de aprendizagem (para pessoas iniciantes aprenderem a mexer e trabalhar na área delas quando não é necessariamente desenvolvimento de app, mas design, ml e etc)
Não acho que o modelo criado já seja utilizado, pois os alinhamentos em geral não são respeitados, mas os elementos em si são criados e identificados muito bem, então o processo do Wireframe já começa com um adianto
Agiliza o processo de criação da interface, facilita a criação de um protótipo, mesmo para quem tem pouco ou nenhum conhecimento do App Inventor.
A ferramenta pode ser muito útil para transferir ideias do papel para o computador. Mesmo não podendo transferir cores imagens ou texto, a ferramenta cria uma base para o App ser desenvolvido.
Acho que ajudará a fazer os apps mais rápido
Porque assim o sistema ajuda a colocar elementos do aplicativo
Foi rápido fazer o desenho e o upload da imagem, parece uma forma interessante de iniciar o desenvolvimento de um app
A ferramenta automatiza o processo te fazer o wireframe da interface
Porque dá para desenvolver um aplicativo mais fácil
Agiliza
Sim, o fato de se poder fazer um esboço no papel e o sistema definir os botões e controles torna o processo intuitivo e relativamente fácil.
É extremamente útil, já que fazer as interfaces no App Inventor é um pesadelo.

Ajuda na criação da interface gráfica do aplicativo abstraindo a parte chata na criação do APP
Sim, pois facilita na hora design da tela
Sim, pois vai facilitar e agilizar o desenvolvimento de apps.
Facilita e agiliza significativamente o processo de transformação de sketch em wireframes. Acredito ser uma ferramenta bastante útil.
Sempre é interessante automatizar um processo. A ferramenta é fácil de ser usada e traz bons resultados.
Facilita o processo de construção de protótipos de aplicativos
Ajuda muito usuários que não tem experiencia em programação, torna o processo de criar apps mais rápido
Esta ferramenta faz com que a criação inicial de um App seja mais rápida, sendo assim o programador pode focar direto aos detalhes e design de seu App em um tempo menor.
E bem fácil de ser usado e atende o necessário
Permite que se trabalhe com a criatividade das pessoas usando as mãos.
Embora não tenha conseguido o resultado esperado, acredito que a ferramenta tem potencial para resolver possíveis erros. Talvez eu tenha feito algo errado, não tenho hábito de fazer wireframes. Recomendo vocês conferirem o resultado.
Apesar de não ser uma ferramenta que eu utilizaria, pois não tenho o costume de criar *sketchs*, vejo muita utilidade na hora de iniciar o projeto quando os *sketchs* já estão criados.
É útil porque acelera o processo de desenvolvimento do aplicativo. Geralmente é mais fácil e demora menos desenhar um sketch do que usar uma ferramenta própria para criar wireframes e, tendo uma ferramenta como o Sketch2aia, que gera um .aia com a maior parte dos componentes presentes já nas posições pensadas, o processo fica ainda mais rápido, pois basta editar as propriedades necessárias.
A ferramenta permitirá uma prototipagem rápida, economizando o tempo de desenvolvedores experientes, e possibilitando também que pessoas com pouco conhecimento se motivem a desenvolver aplicativos.
Incentiva o uso de *sketchs* e agiliza um processo puramente mecânico de posicionar os componentes após fazer o sketch

Foram destacadas como principais vantagens do sistema sua praticidade, facilidade de uso e potencial em economia de tempo no processo de desenvolvimento de um App. A Tabela 22 mostra os principais pontos positivos da ferramenta de acordo com os participantes.

Tabela 22 – Principais pontos positivos da ferramenta

O que mais você gostou da ferramenta Sketch2aia?
Além do fato de passar o sketch para um projeto App Inventor, achei interessante o feedback visual mostrando os componentes identificados nas imagens.
Geração automatizada do arquivo AIA

A ideia e a facilidade de utilizar!
Realmente muito fácil de usar
A facilidade de usar e a agilidade para criar um Wireframe.
A ferramenta é simples e fácil de usar, além de ser fácil de transferir para o App Inventor.
A rapidez para gerar o código
Que ele entendeu os meus desenhos
É um jeito prático de iniciar o desenvolvimento de um app
A possibilidade de automatizar um processo que pode ser bastante demorado.
Praticidade (2)
Simplicidade (4)
Intuitividade
Facilidade (5)
legal
Interface bonito e intuitivo de usar. O resultado sai muito rápido. Parece que o sistema não está reconhecendo o combobox e o appbar.
facilidade em usar e os bons resultados que a ferramenta trouxe
Facilidade e praticidade que ela apresenta para usuários do App Inventor.
Achei fantástica a ideia de tirar do papel uma interface.
A interface de usuário é agradável, o fluxo de navegação é simples e direto, o que economiza tempo.
Interface simples e direta, ótimos resultados.
A identificação bastante precisa dos componentes

Quanto a possíveis oportunidades de melhoria, os voluntários destacaram os layouts dos componentes no *wireframe* de App Inventor gerado pela ferramenta, algo dificultado pelas características internas do próprio App Inventor. A Tabela 23 mostra as sugestões de melhorias dadas pelos participantes do estudo.

Tabela 23 - Sugestões de melhorias dos participantes do estudo

Alguma sugestão de melhoria referente a ferramenta Sketch2aia?
No momento, a organização e posicionamento dos componentes está bem diferente do desenhado nos sketches. Acho que isso pode ter um impacto negativo na avaliação da ferramenta. Como essa organização é uma das partes mais chatas de se fazer no App Inventor, seria bom melhorar nesse

sentido, de forma que o posicionamento dos componentes e espaçamentos desenhados no sketch "batam" com os posicionamentos e espaçamentos no .aia gerado. Uma outra questão que notei no sketch que enviei foi a identificação incorreta de vários labels como sendo switch (p.ex. na tela 3). Seria interessante melhorar essa parte também.
Poderia tentar deixar os componentes alinhados conforme os sketches (para mim eles ficaram todos "sem formatação" na tela), mas nada que atrapalhe o desempenho
Questão do alinhamento dos elementos, acho bem importante
Ajustar o reconhecimento de elementos de diferentes classes quando estes elementos estão em proximidade, o sistema parece agrupar ambos os elementos em uma classe.
Colocar algum anúncio de que para transferir mais de uma imagem para o aplicativo Sketch2aia deve selecionar todas de uma vez.
Seria interessante aceitar figuras png e pdf
Não (6)
Ajustar melhor o tamanho e posição dos elementos
Não está claro o que deve ser inserido nas caixas de entrada da tela de verificação do sketch, principalmente a opção "Deseja usar EscolheLista ou ListaSuspensa?"
Vide comentário do menu para baixar o .aia que está muito difícil de achar. O link do site do appinventor também não está óbvio, deveria ter o link na interface do SDketch2AIA.
Sei que a ferramenta se destina a celular, mas poder usar notebook ou outro dispositivo para gerar a interface, para mim fica mais fácil. Não acho a interação com o celular ergonômica.
talvez um pequeno botão de ajuda com tutoriais de como converter sua imagem em JPEG se não for, como achar a imagem dentro do seu computador caso usar alguns programas comuns, etc... para pessoas que tem pouco conhecimento técnico
Criação de Layouts para desenvolvimento web
na 1º tela, há um erro de digitação "dos interfaces"
Melhorar o reconhecimento de combobox e o appbar
talvez o manual de "como desenhar sketches" deveria ter a leitura "obrigatória" ou pelo menos que o usuário tivesse que acordar ciência de que tem um padrão
Poderia ter a opção de girar a imagem. Foi a única operação adicional que tive que fazer antes de gerar o aia
Escolher uma paleta de cores melhor para identificar antes de realizar o download do .aia.
A princípio conferir se a minha importação foi erro do usuário ou do sistema.
A única coisa que me vem à mente para comentar é o posicionamento dos componentes na tela. No .aia gerado, pelo que vi, os componentes, tendem a ficar no centro, o que nem sempre corresponde ao posicionamento desenhado.
Adicionaria reconhecimento de textos e criaria os labels e botões com os textos escritos nos esboços.
Identificar e incluir de alguma forma no .aia o espaçamento entre os componentes

Algumas das sugestões, como a possibilidade de utilizar formatos de imagens variados, melhorias nas instruções e cores da imagem de *preview* foram aceitas e implementadas desde a realização do estudo.

6. DISCUSSÃO

De forma geral a abordagem se demonstrou precisa na detecção de objetos quanto à geração de *wireframes* em relação à similaridade visual com os *sketches*.

Considerando que a avaliação de performance da detecção de objetos indicou um F1 *score* de 0.88, pode-se concluir que a abordagem geralmente identifica os componentes de UI em *sketches* corretamente. Com a exceção de *ListPicker*, todos os tipos de componentes obtiveram AP acima de 80%. A performance abaixo da média obtida com *ListPicker* se dá provavelmente devido a sua baixa frequência de aparição no conjunto de dados e sua similaridade com outro componentes mais frequentes, como *Button* e *TextBox*.

Futuramente, a introdução de mais exemplos de *ListPicker* no conjunto de dados, ou até mesmo a criação de um padrão diferente com características mais distintas para sua representação em *sketches* podem ajudar a solucionar estes problemas. Comparando esses resultados com o estado da arte também se evidencia a qualidade da abordagem por demonstrar valores de desempenho acima dos trabalhos relacionados.

Os resultados do estudo de usuário também indicam que os *wireframes* gerados pela ferramenta correspondem largamente aos *sketches* utilizados. A maior parte das questões indicadas se referem ao *layout* das UI geradas, como por exemplo, o alinhamento dos elementos de UI (Figura 68).

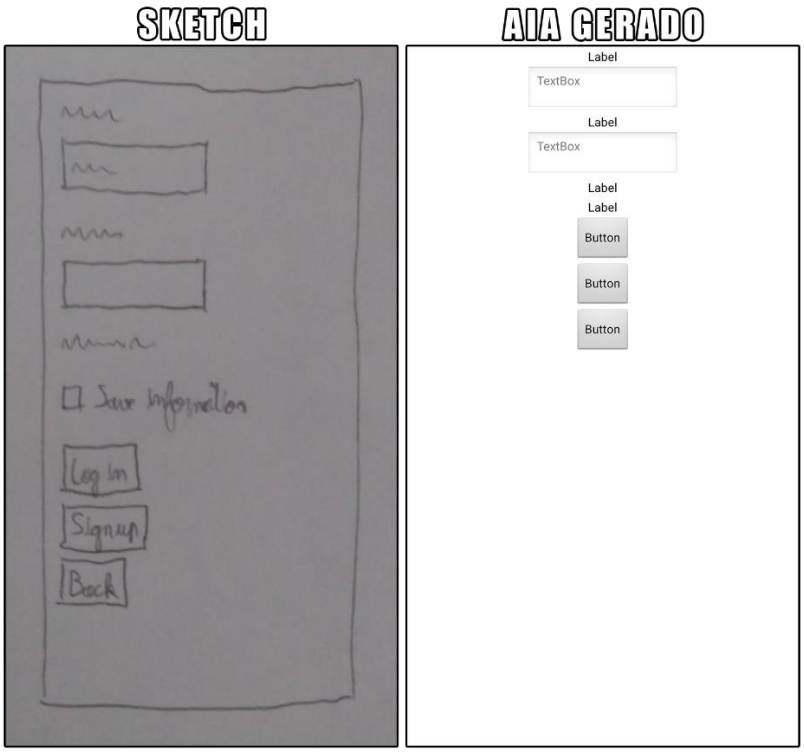


Figura 68 - Exemplo com problema de alinhamento e espaçamento

Também foram observados problemas com espaçamento entre componentes, com a inclusão de espaços em branco sendo uma possível solução para melhor aproximação do *layout* do *sketch*.

Testes de performance também indicam que o sistema pode ser uma maneira eficiente de reduzir o tempo utilizado para design de UI, considerando que um desenvolvedor de UI/UX pode levar em média mais de 1 hora para criar um protótipo de alta fidelidade (BELTRAMELLI 2019), enquanto a geração automática de código de *wireframe* baseada em *sketches* de nossa abordagem leva cerca de 57.47 segundos (para um aplicativo com três telas).

Comparado com outras abordagens, como, por exemplo, REDRAW (MORAN et al., 2018), nossa abordagem é capaz de prototipar várias telas (no máximo 6) de uma aplicação ao mesmo tempo, eliminando a necessidade de prototipar individualmente cada tela e, em seguida, combinar manualmente em uma única aplicação.

Sketch2aia também é capaz de detectar um conjunto de nove componentes de interface do usuário mais utilizados em aplicativos de App Inventor, superando conjuntos muito reduzidos, que limitam algumas abordagens relacionadas (AŞIROĞLU et al. 2019) (ROBINSON 2018) (GE 2019).

Em vez de criar código para ambientes de desenvolvimento móvel profissional, a geração de código do App Inventor também fornece um suporte inexistente até nesse momento para o desenvolvimento de aplicativos móveis pelo usuário final, bem como o contexto de ensino da computação usando o ambiente de programação baseado em blocos App Inventor, que permite a qualquer pessoa criar aplicativos móveis, projetando a interface com um simples desenho.

Ameaças à validade. Devido às características desse tipo de pesquisa, identificamos possíveis ameaças e aplicamos estratégias de mitigação para minimizar seu impacto. Para criar um conjunto de dados autêntico e variado, criamos *sketches* manualmente para as UIs de aplicativos App Inventor, envolvendo uma amostra de 28 participantes, reduzindo o risco de *sketches* que não representam adequadamente *sketches* realistas desenhados por um público-alvo com diversas origens e idades. O tamanho da amostra de 279 *sketches* de UI de aplicativos do App Inventor selecionados aleatoriamente na Galeria do App Inventor e de aplicativos desenvolvidos pela iniciativa Computação na Escola fornecem uma amostra representativa das UIs de aplicativos desenvolvidos com App Inventor.

A seleção do modelo de *deep learning* foi feita sistematicamente com base na literatura como resultado de testes informais, por exemplo, comparando diferentes versões do YOLO para maximizar os resultados de desempenho. A seleção das medidas de desempenho também foi realizada com base na literatura que indica as

medidas comuns utilizadas para esse tipo de aplicação, a fim de assegurar uma avaliação adequada.

Com relação ao estudo de avaliação, definimos sistematicamente o estudo usando a abordagem GQM (BASILI et al., 1994). Em comparação com as avaliações em trabalhos relacionados, também consideramos aceitável a comparação de um total de 11 *sketches* com os *wireframes* gerados, para fornecer as primeiras ideias sobre a correspondência dos resultados. Com relação ao tamanho da amostra, nossa avaliação utilizou dados coletados de 29 participantes. No contexto de uma avaliação preliminar, esse é um tamanho de amostra aceitável que permite obter os primeiros resultados. No entanto, como os dados foram obtidos em apenas um contexto, serão necessários mais estudos para aumentar a validade externa.

7. CONCLUSÃO

O objetivo principal deste trabalho era o desenvolvimento de um modelo de geração automática de *wireframes* no *App Inventor* a partir de *sketches* usando d*eep learning*. Para a realização deste objetivo, foi realizada inicialmente uma análise da fundamentação teórica sobre aprendizagem de elementos de *design* de interface de usuário e *deep learning* (O1). Também foi levantado o estado da arte em relação a automação da conversão de *sketches* em *wireframes* de apps Android (O2). Esta análise levantou diversos avanços na detecção de elementos de interface e na geração automática de *wireframes*, contudo identificou-se a ausência de modelos e ferramentas específicas para o App Inventor. O modelo de detecção de componentes de interface foi desenvolvido e testado com técnicas de *deep learning* utilizando o framework Darknet (O3). Observando desempenho satisfatório do modelo de detecção de componentes de interface, também foi desenvolvido o modelo de geração automática de códigos de wireframe de App Inventor (O4). O resultado da pesquisa é disponibilizado como uma ferramenta web, disponibilizada online no endereço: http://www.gqs.ufsc.br/sketch2aia/.

Em termos de impacto científico, é criado um conhecimento pioneiro e inovador em relação a geração automática de *wireframes* de apps Android a partir de *sketches*. É criado também um modelo inovador que permite a geração automática de *wireframes* para App Inventor a partir de *sketches*. Por fim, é criado e disponibilizado também um conjunto de dados com *sketches* feitos a mão e fotografados em condições realistas. Em termos tecnológicos foi criado um sistema web disponível online para facilitar o processo de design de wireframes. Em termos sociais, com a disponibilidade do modelo desenvolvido, espera-se contribuir com a melhoria do

ensino de computação no âmbito do ensino brasileiro com relação a aprendizagem e popularização da computação e desenvolvimento de apps na sociedade.

Como trabalhos futuros, sugere-se realizar uma análise mais ampla do conjunto de dados e do próprio modelo, visando melhorar e adaptar ambos com base nos erros e oportunidades de melhoria atualmente observados. Recomenda-se também desenvolver um modelo melhor para geração de código de wireframe a partir dos elementos detectados, com *layouts* mais similares aos observados nos *sketches*. Por fim, recomenda-se também a inclusão de reconhecimento de texto na ferramenta, possibilitando o preenchimento automático de componentes.

Referências

ALVES, N. da C.; GRESSE VON WANGENHEIM, C.; HAUCK, J. C. R. **Teaching Programming to Novices: A Large-scale Analysis of App Inventor Projects.** In: Proc. of the XLVI Conferencia Latinoamericana de Informática, Ecuador, 2020.

AppInventor.org (2019). **Course In A Box**. Disponível em: <http://www.appinventor.org/content/CourseInABox/Intro>. Acesso em: novembro 2019.

AŞIROĞLU, B.; METE, B. R.; YILDIZ, E.; NALÇAKAN, Y.; SEZEN, A.; DAĞTEKIN, M.; ENSARI, T. **Automatic HTML Code Generation from Mock-up Images Using Machine Learning Techniques**. in *Proc. of the Scientific Meeting on Electrical-Electronics and Biomedical Engineering and Computer Science*, 2019, pp. 1-4.

BANGOR, A.; KORTUM, P.; MILLER, J.; **Determining what individual SUS scores mean: adding an adjective rating scale**. J. Usability Studies 4, 3, 114–123. 2009.

BASILI, V. R. et al. **Goal, Question Metric Paradigm**. In J. J. Marciniak, Encyclopedia of Software Engineering, Wiley-Interscience, New York. 1994.

BELTRAMELLI, T. **pix2code: Generating Code from a Graphical User Interface Screenshot**. In *Proc. of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2018, pp. 1–6.

BELTRAMELLI, T. **Teaching Machines to Understand User Interfaces**, 2017. Disponível em: <https://hackernoon.com/teaching-machines-to-understand-user-interfaces-5a0cdeb4d579> Acesso em: novembro 2019.

BLOCKLY, 2018. Disponível em: https://developers.google.com/blockly/. Acesso em: outubro 2020.

BOCHKOVSKIY, A.; **Yolo-v3 and Yolo-v2 for Windows and Linux**; 2019. Disponível em: <https://github.com/AlexeyAB/darknet/>. Acesso em: maio 2020.

BUSHAEV, V.; **How do we 'train' neural networks?**, 2017. Disponível em: <https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94>. Acesso em: novembro 2019.

CHEN, L.; ZHU, Y.; PAPANDREOU, G.; SCHROFF, F.; ADAM, H. **Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation**, 2017. Disponível em: < https://arxiv.org/abs/1802.02611>. Acesso em: novembro 2019.

CHOLLET, F. **Xception: Deep Learning with Depthwise Separable Convolutions**, 2017. Disponível em: <https://arxiv.org/abs/1610.02357>. Acesso em: novembro 2019.

CSTA. ACM. **CSTA K –12 Computer Science Standards**, 2016. Disponível em: <https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf>. Acesso em: outubro 2020.

Fastai; **About**. Disponível em: <https://www.fast.ai/about/>. Acesso em: outubro 2020.

FU, B.; LIN, J.; LI, L.; FALOUTSOS, C.; HONG, J.; SADEH, N. **Why people hate your app: making sense of user feedback in a mobile app store**. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '13), Chicago, Illinois, USA, 2013.

GE, X. **Android GUI search using hand-drawn sketches**. In: Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE '19), Piscataway, USA. 2019.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A.; **Deep Learning**. Cambridge, MA: MIT Press, 2016.

GQS, 2019. **Conheça o app para visualizar pontos de coleta do ECOPET na Grande Florianópolis!**. Disponível em: <https://ine.ufsc.br/2019/07/12/conheca-o-app-para-visualizar-pontos-de-coleta-do-ecopet-na-grande-florianopolis/>. Acesso em: novembro 2019.

GRESSE VON WANGENHEIM, C. HAUCK, J. C. R.; DA CRUZ PINHEIRO, F.; BARBOSA, H.; ARAUJO PORTO, J. V.; DINIZ DA SILVEIRA, T.; CORREA, O. A. **Documentação Técnica Aplicativo ClicDenúncia.** Relatório técnico INCoD/GQS.06.2017.P.

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

HOWARD, J.; **fastai v1 for PyTorch: Fast and accurate neural nets using modern best practices**, 2018. Disponível em: <https://www.fast.ai/2018/10/02/fastai-ai/>. Acesso em: outubro 2019.

HUANG, F; F. CANNY, J; NICHOLS, J. **Swire: Sketch-based User Interface Retrieval**. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, Glascow, Escócia, 2019.

INDIA, U.; Difference between Machine Learning, Deep Learning and Artificial Intelligence, 2018. Disponível em: <https://medium.com/@UdacityINDIA/difference-between-machine-learning-deep-learningand-artificial-intelligence-e9073d43a4c3>. Acesso em: outubro 2020.

KARKARE, P.; **Convolutional Neural Networks – Simplified**, 2019. Disponível em: <https://medium.com/x8-the-ai-community/cnn-9c5e63703c3f>. Acesso em: novembro 2019.

KAS, A. **UI/UX sketching techniques 101**, 2019. Disponível em: <https://uxdesign.cc/ui-ux-sketching-techniques-101-7e91d854ae3d>. Acesso em: novembro 2019.

KIERSKI, M. **Machine Learning development process - you've got it wrong**, 2017. Disponível em: <https://medium.com/sigmoidal/machine-learning-development-process-youve-got-it-wrong-396270e653f4>. Acesso em: novembro 2019.

KIM, B.; PARK, S.; WON, T.; HEO, J.; KIM, B. **Deep-Learning Based Web UI Automatic Programming**. In: Proceedings of International Conference on Research in Adaptive and Convergent Systems, Honolulu, USA, 2018.

Landay, J.; Myers, B. (1994) **Interactive Sketching for the Early Stages of User Interface Design.** Disponível em: <https://apps.dtic.mil/dtic/tr/fulltext/u2/a285339.pdf>. Acesso em: novembro 2019.

LECUN, Y.; BENGIO, Y.; HINTON, G. **Deep Learning**. Nature. 521. 436-44, 2015.

LIU, Y.; HU, Q.; SHU, K.; **Improving pix2code based Bi-directional LSTM**. In: IEEE International Conference on Automation, Electronics and Electrical Engineering (AUTEEE), Shenyang, China, 2018.

MISSFELDT FILHO, R.; AZEVEDO, L. F.; GRESSE VON WANGENHEIM, C. **Documentação Técnica Aplicativo Bikanto.** Relatório Técnico INCoD/INE/UFSC INCoD/GQS.16.2017.P.

MIT App Inventor. **About Us**, 2019. Disponível em: <http://appinventor.mit.edu/explore/about-us.html>. Acesso em: Novembro 2019.

MITCHELL, T. M; **Machine Learning**, 1ª edição, Nova Iorque, NY: McGraw-Hill Education, 1997

MORAN, K. P.; BERNAL-CARDENAS, C.; CURCIO, M.; BONETT, R.; POSHYVANYK, D. **Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps**. In: in IEEE Transactions on Software Engineering, 2018.

MUSTAFARAJ, E. et al. 2017. **Identifying original projects in App Inventor**. In *Proc. of the 30th Int. Flairs Conference*, Marco Island, USA. 2017.

NEIL, T. **Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps**. O'Reilly Media Inc., Sebastopol, CA, USA, 2014.

NGUYEN, T. A.; Christoph CSALLNER, C. **Reverse engineering mobile application user interfaces with REMAUI**. In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, Piscataway, USA, 2015.

PANT, A.; **Workflow of a Machine Learning Project**, 2019. Disponível em: <https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94>. Acesso em: novembro 2019.

PASTERNAK, E.; FENICHEL, R.; MARSHALL, A. N. **Tips for Creating a Block Language with Blockly**. IEEE Blocks and Beyond Workshop. Raleigh, NC, USA. 2017.

Pathmind, **A Beginner's Guide to Convolutional Neural Networks (CNNs)**. Disponível em: <https://wiki.pathmind.com/convolutional-network>. Acesso em: novembro 2019a.

Pathmind, **A Beginner's Guide to Neural Networks and Deep Learning**. Disponível em: <https://wiki.pathmind.com/neural-network>. Acesso em: novembro 2019b.

Patton, E.; Tissenbaum, M.; Harunani, F. (2019). **MIT App Inventor: Objectives, Design, and Development.** Disponível em: <https://link.springer.com/content/pdf/10.1007%2F978-981-13-6528-7_3.pdf>. Acesso em: novembro 2019.

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

PIZZANI, L.; SILVA, R. C. DA; BELLO, S. F.; HAYASHI, M. C. P. I. **A arte da pesquisa bibliográfica na busca do conhecimento**. RDBCI: Revista Digital de Biblioteconomia e Ciência da Informação, v. 10, n. 2, p. 53-66, 10 jul. 2012.

POLYZOTIS, N.; ROY, S.; WHANG, S. E.; ZIENKEVICH, M. **Data Management in Production Machine Learning**. Proceedings of the ACM International Conference on Management of Data, Chicago, IL, USA, 2017.

PRESSMAN, R. **Engenharia de software: Uma abordagem profissional**. 8a edição. Porto Alegre: Bookman, 2016. 968 p.

REDMON, J.; **Darknet: Open Source Neural Networks in C;** 2016. Disponível em: <http://pjreddie.com/darknet/>. Acesso em: maio 2020.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A.: **You Only Look Once: Unified, Real-Time Object Detection**, 2015. Disponível em: <https://arxiv.org/abs/1506.02640>. Acesso em: outubro 2020.

REDMON, J.; FARHADI, A. **YOLOv3: An Incremental Improvement**, 2019. Disponível em: <https://arxiv.org/abs/1804.02767>. Acesso em: novembro de 2019.

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

ROBINSON, A. **Sketch2code: Generating a website from a paper mockup**. Dissertation, University of Bristol, UK, 2019.

ROGERS, Y.; SHARP, H; PREECE, J. **Design de Interação: Além da Interação Homem-Computador,** 3ª edição, Porto Alegre, RS, Brasil, Bookman Editora, 2013.

RUSSAKOVSKY, O. et al. **ImageNet Large Scale Visual Recognition Challenge**. arXiv:1409.0575 [cs.CV], 2014.

RUSSEL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**, 3ª edição. New Jersey, NY, USA, Prentice Hall, 2009.

SCHLATTER, T.; LEVINSON, D. **Visual Usability: Principles and practices for designing digital applications**. 2013.

Scratch. Disponível em: <https://scratch.mit.edu>. Acesso em: novembro 2019.

SHUAI. **Steps to Follow in Deep Learning Projects**, 2017. Disponível em: <https://shuaiw.github.io/2017/09/27/steps-to-follow-in-deep-learning-projects.html>. Acesso em: novembro 2019.

SNAP!. Disponível em: <https://snap.berkeley.edu/>. Acesso em: novembro 2019.

SULERI, S.; PANDIAN, V. P. S.; SHISHKOVETS, S.; JARKE, M. **Eve: A Sketch-based Software Prototyping Workbench**," in Proc. of the Conference on Human Factors in Computing Systems, 2019, pp. 1–6.

VON WANGENHEIM, A.; **Visão Computacional**: Deep Learning, 2018. Disponível em: <http://www.lapix.ufsc.br/ensino/visao-computacional/visao-computacionaldeep-learning>. Acesso em: novembro 2019.

WEISSTEIN, E.; **Convolution**. MathWorld -- A Wolfram Web Resource. Disponível em: <http://mathworld.wolfram.com/Convolution.html>. Acesso em: novembro 2019.

WOLBER, D.; ABELSON, H.; FRIEDMAN, M. **Democratizing Computing with App Inventor**, 2015. Disponível em: <https://doi.org/10.1145/2721914.2721935>. Acesso em: novembro 2019.

YOSINSKI, J. et al. **How transferable are features in deep neural networks?**. In Proc. of the 27th Int. Conference on Neural Information Processing Systems, Montreal, Canada, 3320-3328, 2014.

YUN, Y; JUNG, J.; EUN, S.; SO, S.; HEO, J. **Detection of GUI elements on sketch images using object detector based on deep neural networks**. 6th International Conference on Green and Human Information Technology, Chiang Mai, Thailand, 2018.

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

APÊNDICE A

Para converter os arquivos JSON gerados pelo programa labelme para o formato TXT utilizado como entrada para o treinamento da rede Darknet, foi criado um pequeno *script* em Python, que pode ser visto na Figura 1.

```
import json
import os
import glob

dataset = './dataset'
outpath = "./result/"

labels = ['Screen','Title', 'Button', 'Text', 'TextField', 'Image', 'List']

for filename in glob.glob(os.path.join(dataset, '*.json')): #only process .JSON
files in folder.
    with open(filename, encoding='utf-8', mode='r') as currentFile:

        data = json.load(currentFile)

        imageWidth = data["imageWidth"]
        imageHeight = data["imageHeight"]

        shapes = data["shapes"]

        dw = 1./imageWidth
        dh = 1./imageHeight

        lines = []

        for shape in shapes:
            labelClass = labels.index(shape['label'])

            x1 = shape['points'][0][0]
            y1 = shape['points'][0][1]
            x2 = shape['points'][1][0]
            y2 = shape['points'][1][1]

            x = (x1 + x2)/2.0
            y = (y1 + y2)/2.0
            w = abs(x1 - x2)
            h = abs(y1 - y2)
            x = x*dw
            w = w*dw
            y = y*dh
            h = h*dh

            lines += [str(labelClass) + ' ' + str(x) + \
                     ' ' + str(y) + ' ' + str(w) + ' ' + str(h) + '\n']

        newfile = filename[:-5] + ".txt"

        with open(newfile, encoding='utf-8', mode='w') as textFile:
            textFile.writelines(lines)
```

Figura 1 – *Script* Python para conversão dos arquivos de *labeling* de JSON para TXT

APÊNDICE B

Para separar o conjunto de dados em um conjunto de treino e um de validação foi criado um pequeno *script* em Python, que pode ser visto na Figura 1.

```
import glob
import os

current_dir = os.getcwd()
split_pct = 15  # 15% validation set
file_train = open(current_dir + "/train.txt", "w")
file_val = open(current_dir + "/val.txt", "w")

counter = 1
index_test = round(100 / split_pct)
for fullpath in glob.iglob(os.path.join(current_dir + '/images', "*.JPG")):
    title, ext = os.path.splitext(os.path.basename(fullpath))
    if counter == index_test:
        counter = 1
        file_val.write(current_dir + "/" + title + '.JPG' + "\n")
    else:
        file_train.write(current_dir + "/" + title + '.JPG' + "\n")
        counter = counter + 1
file_train.close()
file_val.close()
```

Figura 1 – *Script* Python para conversão dos arquivos de *labeling* de JSON para TXT

APÊNDICE C

Questionário utilizado para a Avaliação Preliminar do sistema Sketch2aia por meio de um Estudo de Usuário.

Avaliação do Sketch2aia (CnE/INCoD/INE/UFSC)

Você está sendo convidado(a) a participar do painel de especialistas para avaliar a ferramenta "Sketch2aia" que visa gerar automaticamente um protótipo (wireframe) de um app no App Inventor a partir de um sketch desenhado a mão. A ferramenta pode ser utilizada tanto no ensino de computação na Educação Básica e Superior quanto por usuários finais interessados em criar apps.

A pesquisa faz parte da iniciativa Computação na Escola do INCoD/INE/UFSC. O objetivo deste painel de especialistas é avaliar a qualidade da ferramenta desenvolvida como resultado do TCC do Daniel Baulé no CCO/INE/UFSC. Como parte da sua participação estaremos solicitando a sua avaliação de 10 protótipos comparando o sketch e o wireframe automaticamente gerado, além de uma geração de um protótipo a partir de qualquer sketch desenhado por você e em seguida responder um questionário de feedback. A sua participação é voluntária sem remuneração. Todos os seus dados serão tratados de forma sigilosa usados somente para fins de pesquisa. Assumimos que a participação deve levar em torno de 30 min.

Como a ferramenta ainda está em fase de teste gostaríamos pedir para ainda não compartilhar o acesso a ferramenta. Logo estaremos disponibilizando a versão final via o site da Computação na Escola.

PRAZO: 22/06/2020 (Favor avisar, se necessite um prazo maior)

Qualquer dúvida favor entre em contato:
Daniel (dsbaule@gmail.com) e Christiane (gresse@gmail.com)

Muito obrigada pela sua participação - ela é muito importante para nos ajudar a melhorar o ensino da computação no futuro!
*Obrigatório

Dados do voluntário

1. Nome *

2. Principal área de conhecimento *

Marcar apenas uma oval.

- ◯ Ciências da Computação
- ◯ Design
- ◯ Ensino de Computação
- ◯ Outro: _____

3. Nível de escolaridade *

Marcar apenas uma oval.

- ◯ Ensino Médio
- ◯ Graduação
- ◯ Mestrado
- ◯ Doutorado
- ◯ Outro: _____

4. Já criou um App com App Inventor? *

Marcar apenas uma oval.

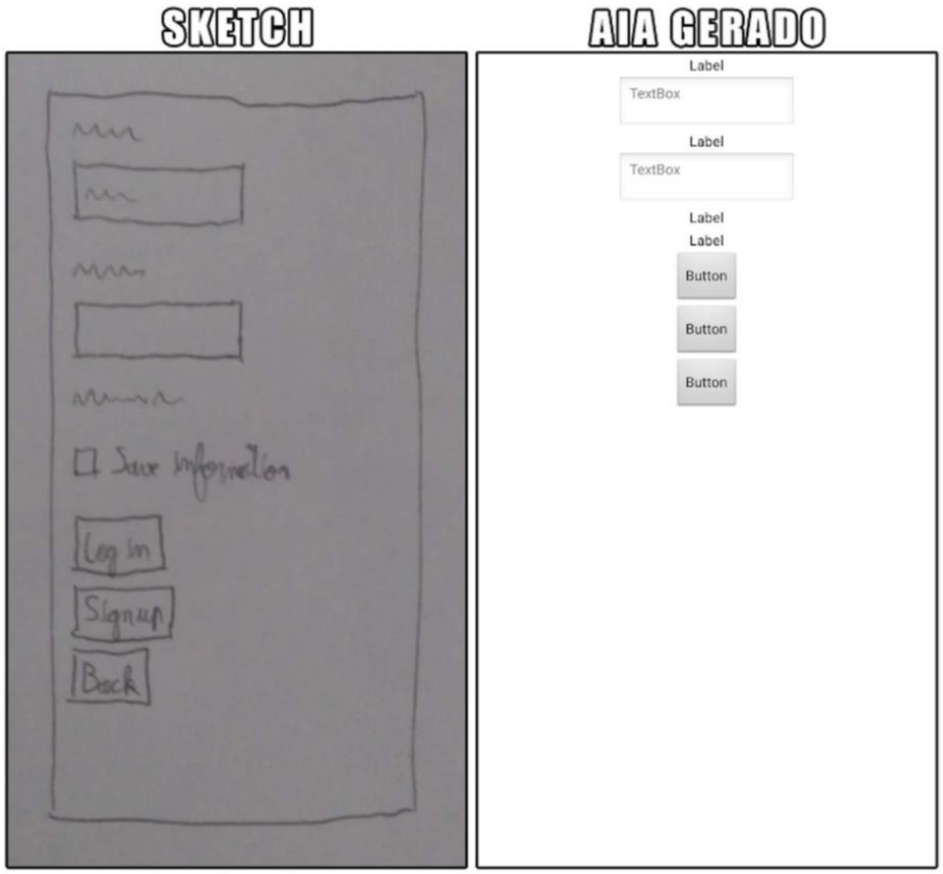
- ◯ Sim
- ◯ Não

Avaliação de Protótipos

O objetivo da ferramenta é gerar protótipos de apps no nível de wireframes (somente elementos de interface de usuário e o seu layout na tela - sem cores, tipografia, imagens, etc.) no App Inventor a partir de um sketch desenhado a mão.

Para cada um dos seguintes protótipos automaticamente gerado com Sketch2aia avalie quanto ele corresponde com o sketch.

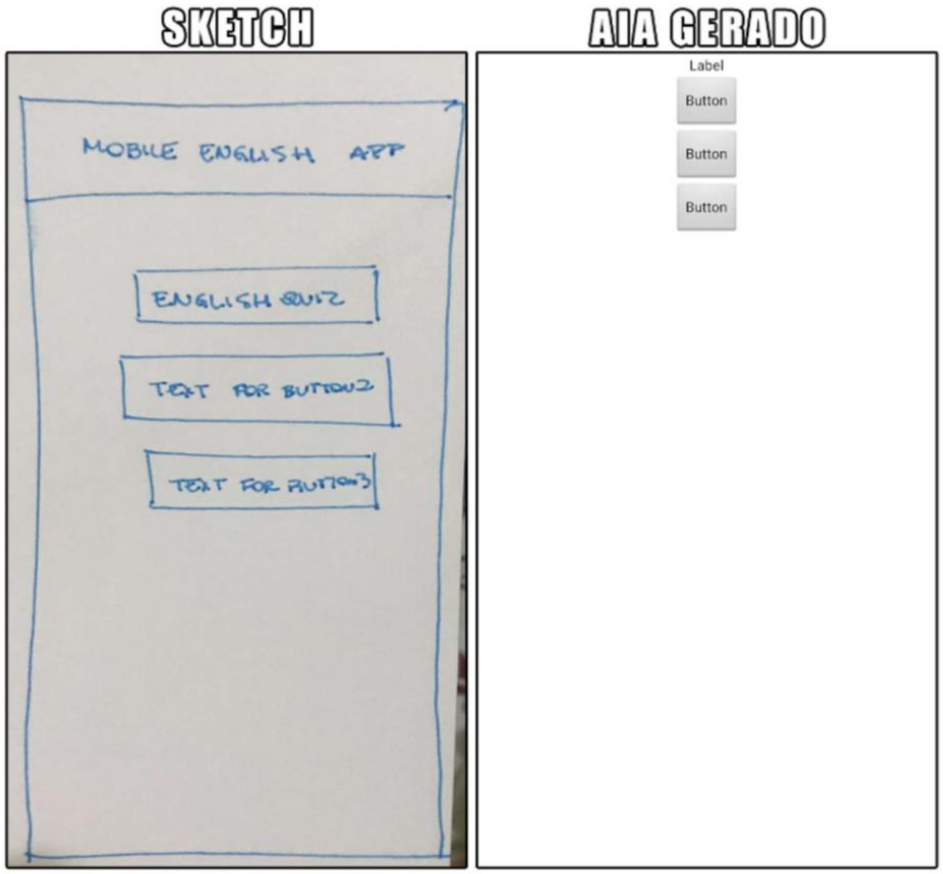
5. IMAGEM DO SKETCH/SCREENSHOT 1 *



Marcar apenas uma oval.

- Corresponde totalmente
- Corresponde largamente
- Corresponde pouco
- Não corresponde

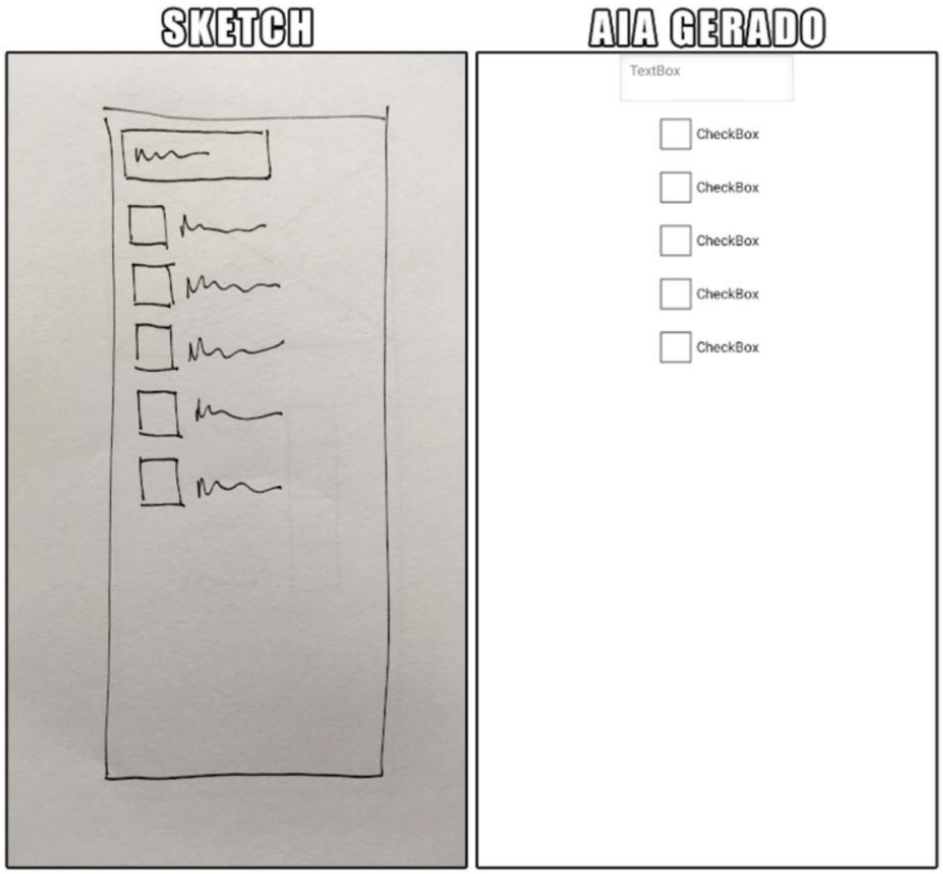
6. Protótipo 2 *



Marcar apenas uma oval.

- ◯ Corresponde totalmente
- ◯ Corresponde largamente
- ◯ Corresponde pouco
- ◯ Não corresponde

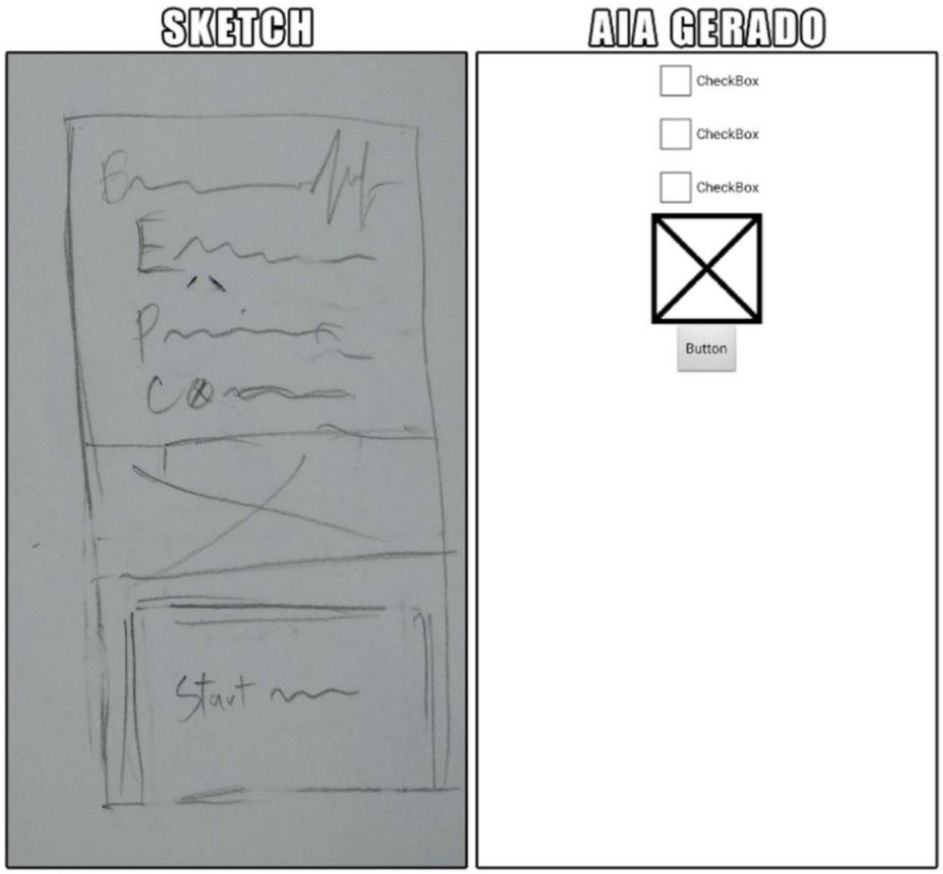
7. Protótipo 3 *



Marcar apenas uma oval.

- Corresponde totalmente
- Corresponde largamente
- Corresponde pouco
- Não corresponde

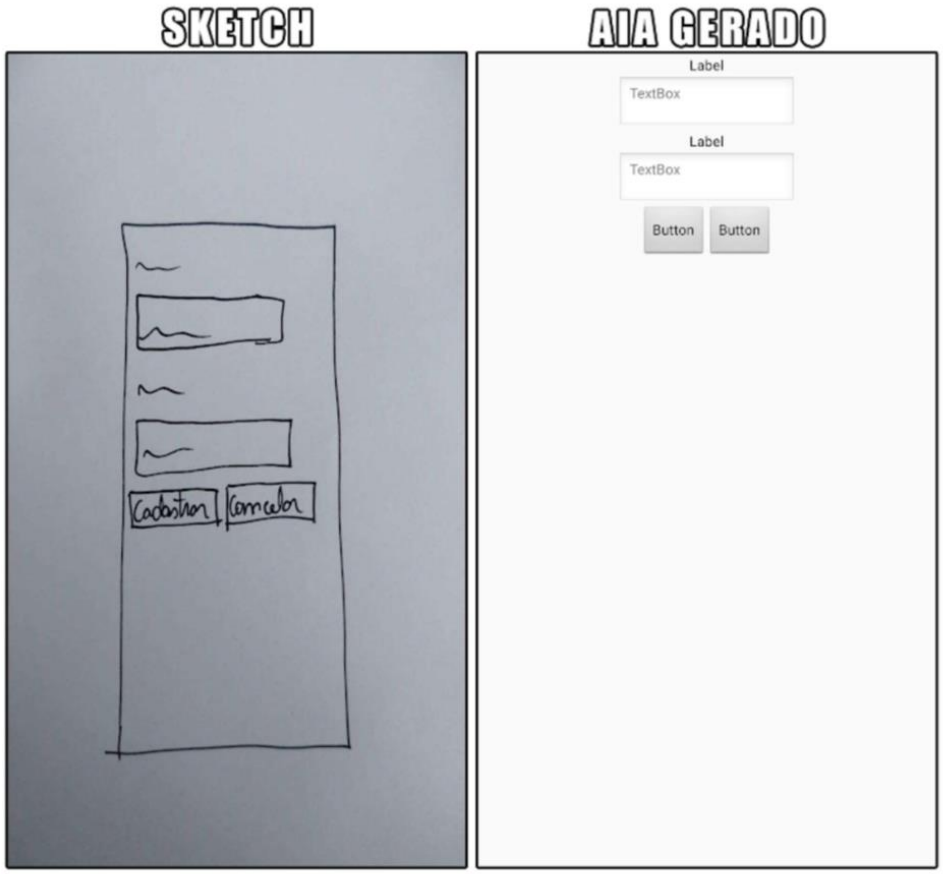
8. Protótipo 4 *



Marcar apenas uma oval.

- Corresponde totalmente
- Corresponde largamente
- Corresponde pouco
- Não corresponde

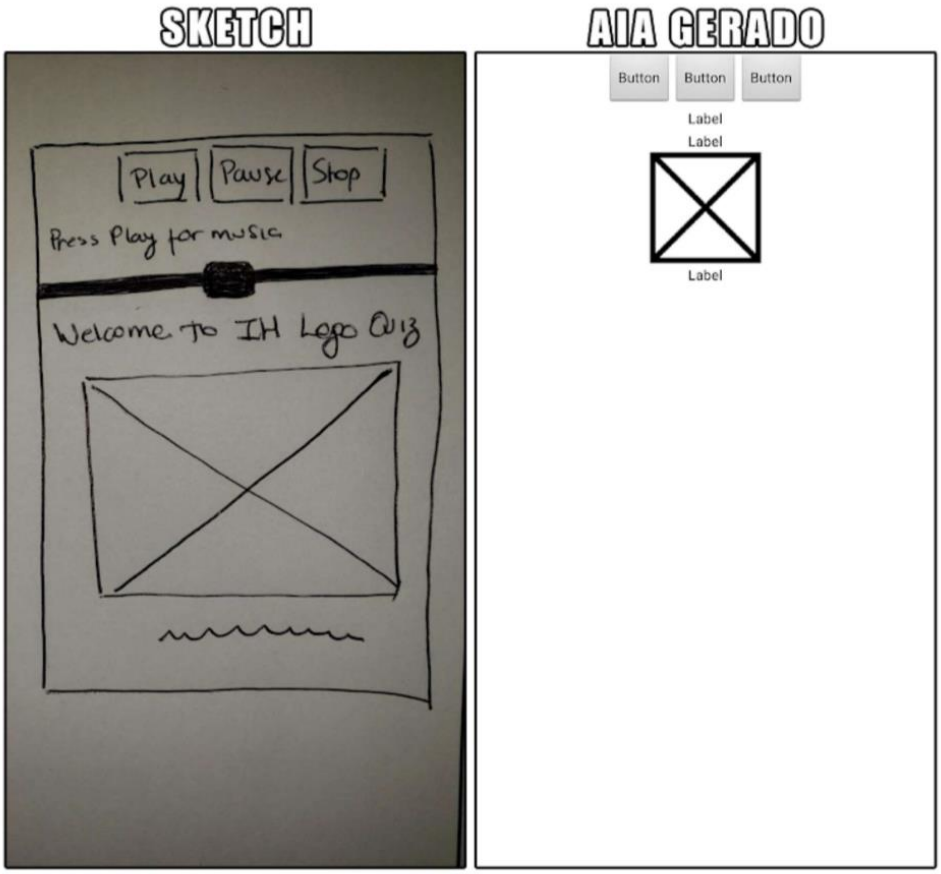
9. Protótipo 5 *



Marcar apenas uma oval.

- ◯ Corresponde totalmente
- ◯ Corresponde largamente
- ◯ Corresponde pouco
- ◯ Não corresponde

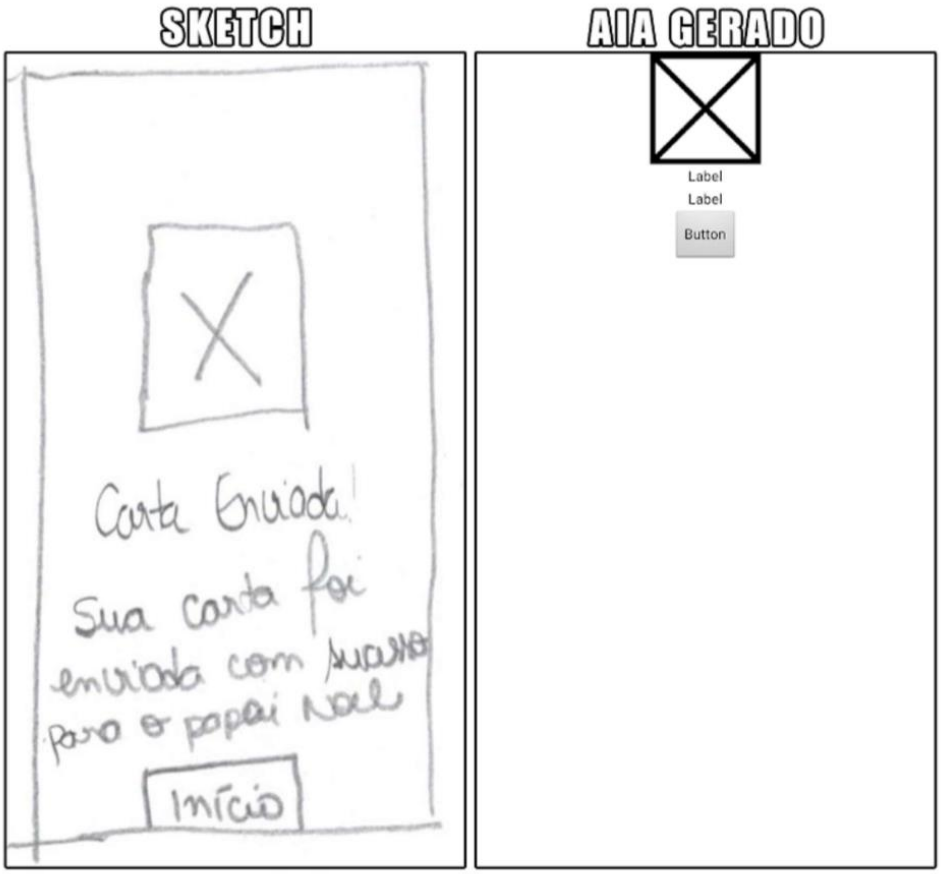
10. Protótipo 6 *



Marcar apenas uma oval.

- ◯ Corresponde totalmente
- ◯ Corresponde largamente
- ◯ Corresponde pouco
- ◯ Não corresponde

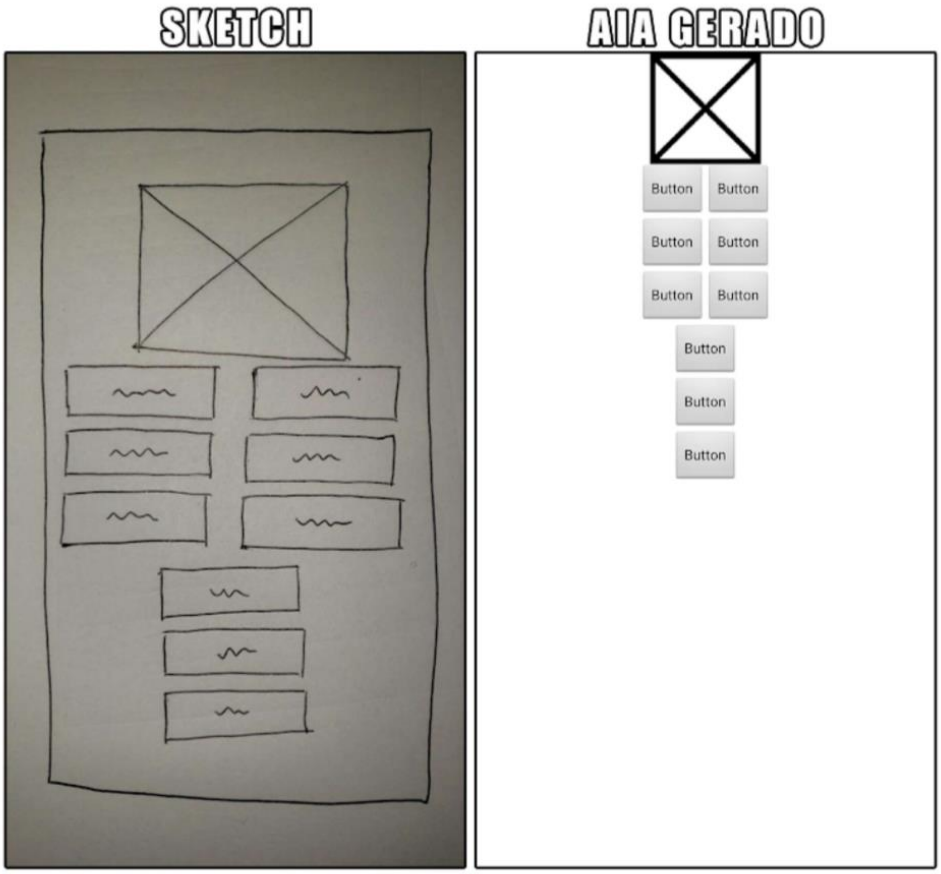
11. Protótipo 7 *



Marcar apenas uma oval.

- ◯ Corresponde totalmente
- ◯ Corresponde largamente
- ◯ Corresponde pouco
- ◯ Não corresponde

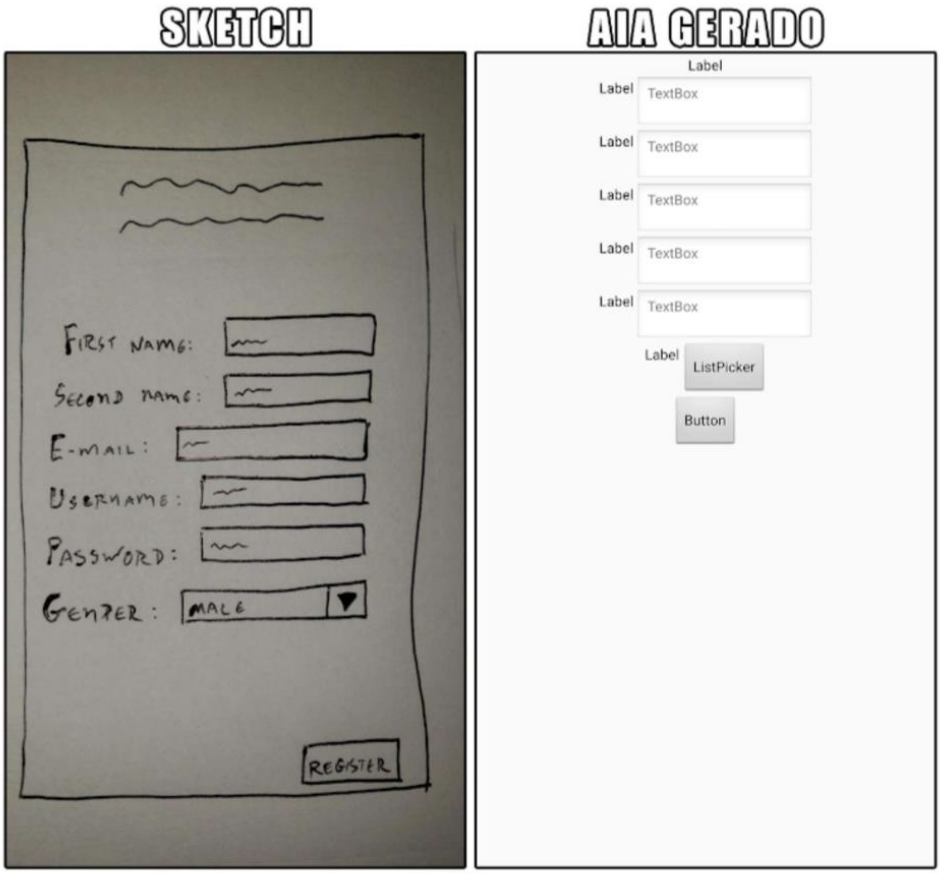
12. Protótipo 8 *



Marcar apenas uma oval.

- ◯ Corresponde totalmente
- ◯ Corresponde largamente
- ◯ Corresponde pouco
- ◯ Não corresponde

13. Protótipo 9 *



Marcar apenas uma oval.

- ◯ Corresponde totalmente
- ◯ Corresponde largamente
- ◯ Corresponde pouco
- ◯ Não corresponde

14. Protótipo 10 *

Marcar apenas uma oval.

- ◯ Corresponde totalmente
- ◯ Corresponde largamente
- ◯ Corresponde pouco
- ◯ Não corresponde

Avaliando com um Sketch qualquer

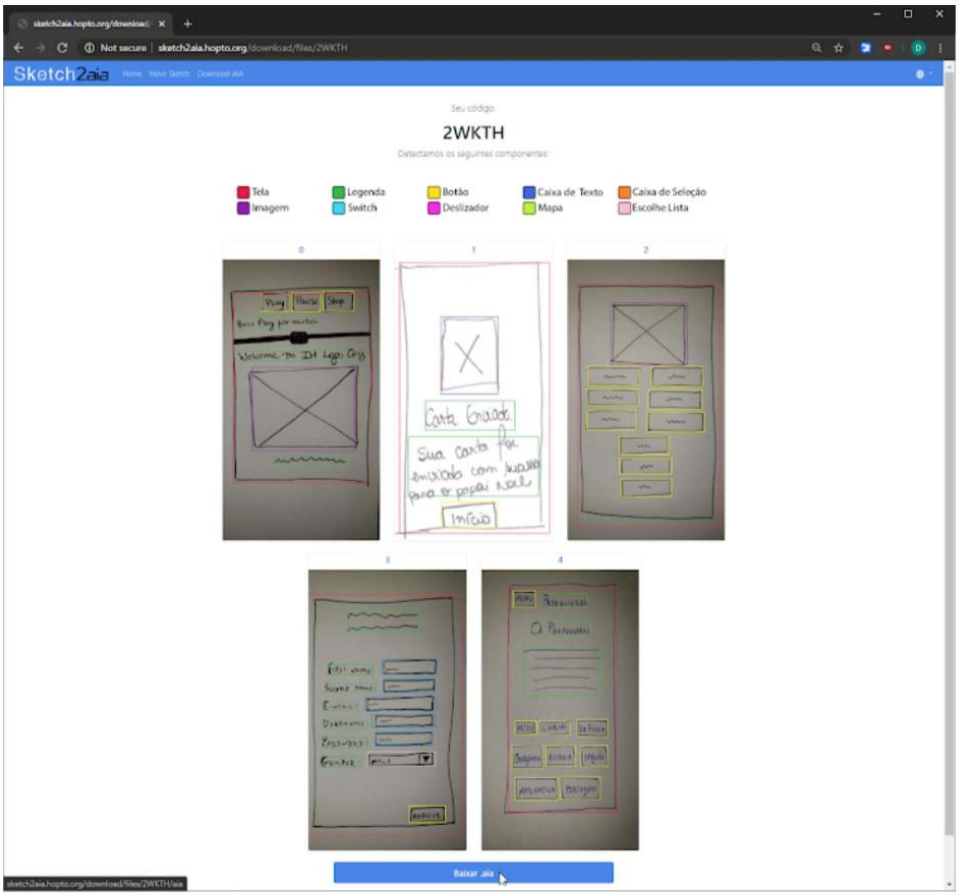
Agora explore a ferramenta livremente usando qualquer sketch sua de uma tela de um aplicativo móvel (Android).

A ferramenta está disponível para teste aqui: http://sketch2aia.hopto.org/

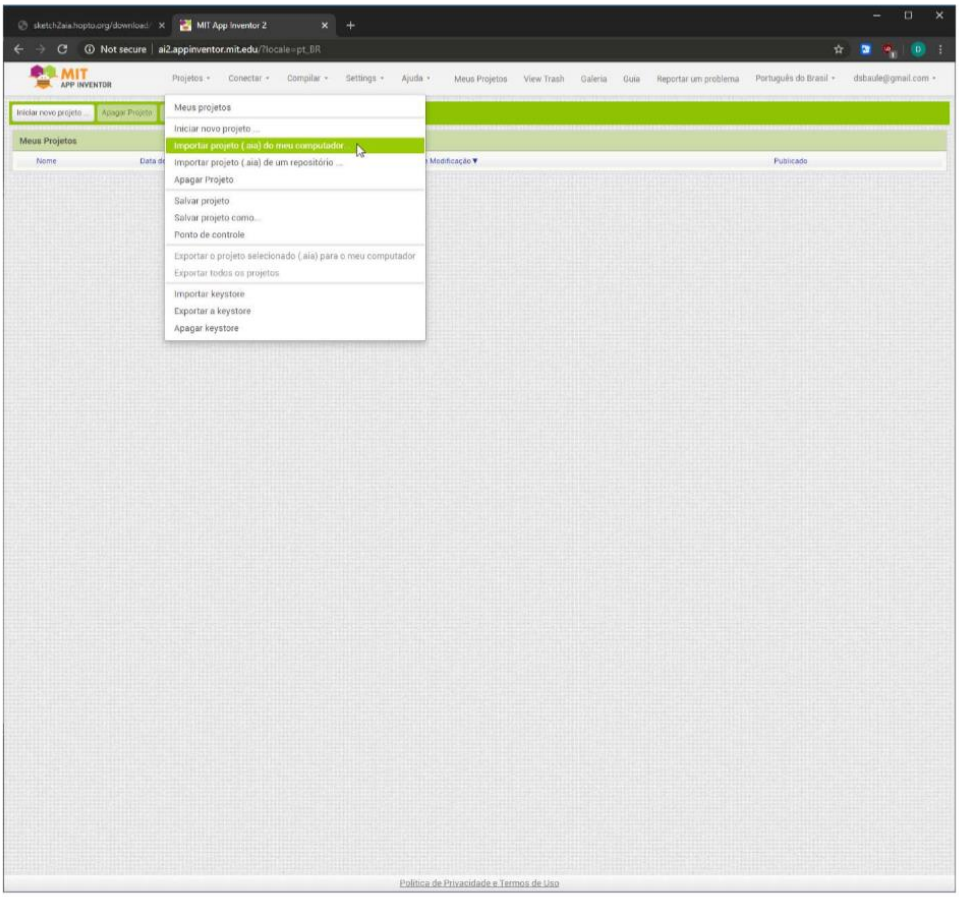
Como verificar o resultado gerado pela ferramenta Sketch2aia

15. Informe aqui o código do seu projeto (informado pela ferramenta, tipo 2WKTH) *

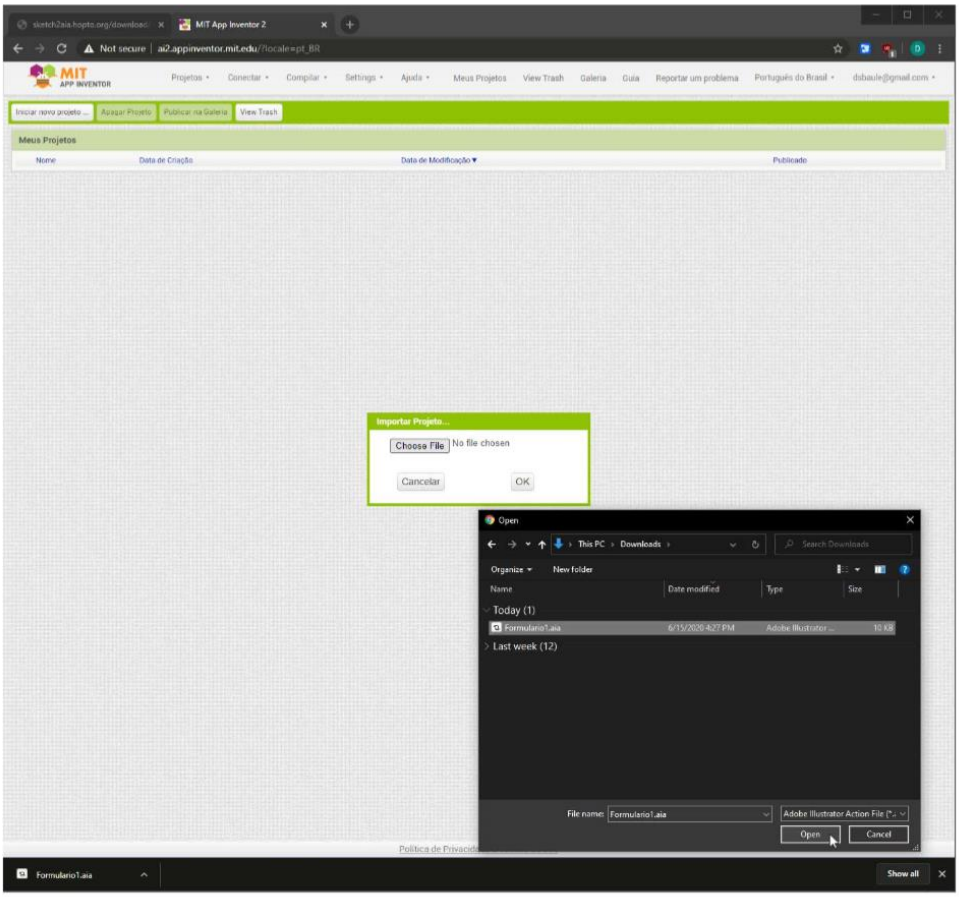
1 - Baixe o arquivo .aia gerado em seu computador



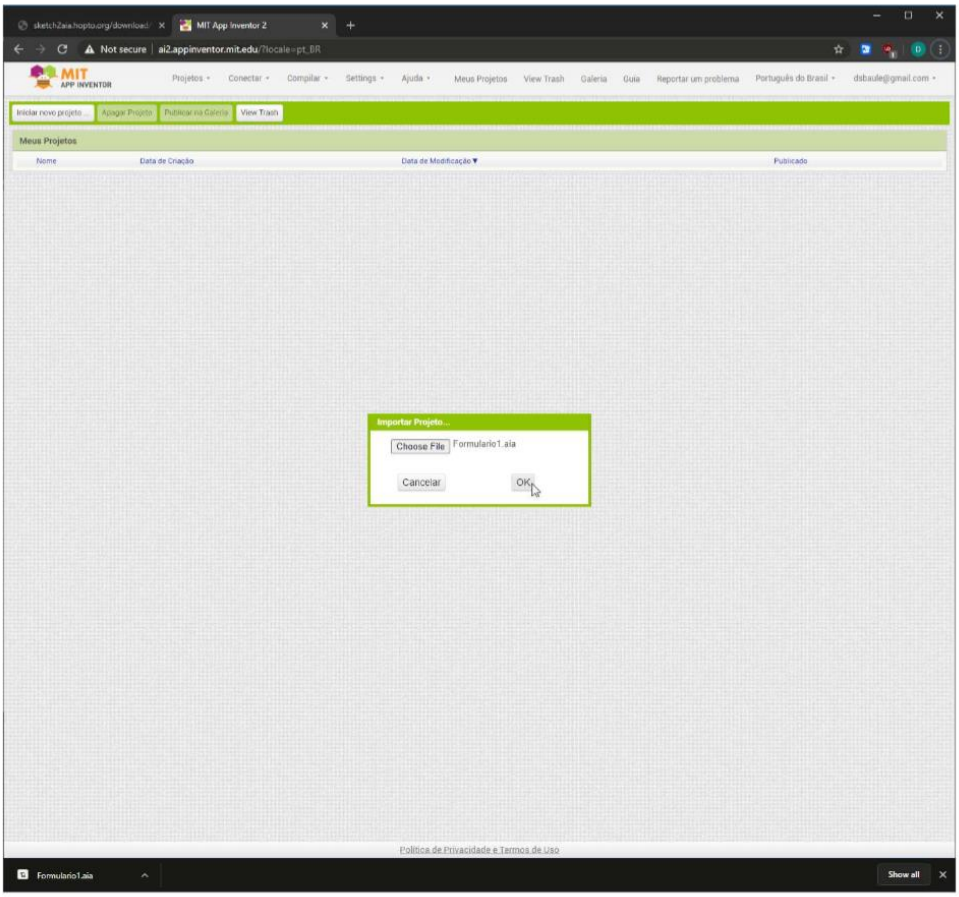
2 - No App inventor, clique em "Projetos > Importar projeto (.aia) do meu computador"



3 - Selecione o arquivo baixado do Sketch2aia



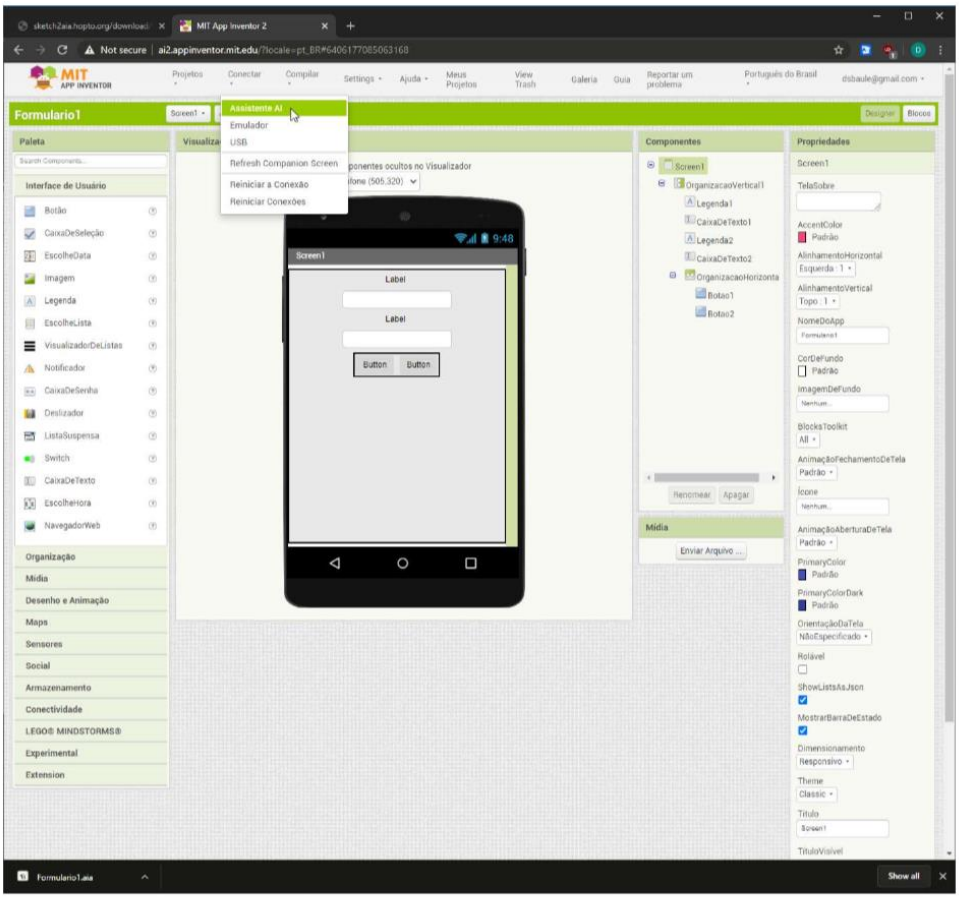
4 - Clique OK para finalizar a importação



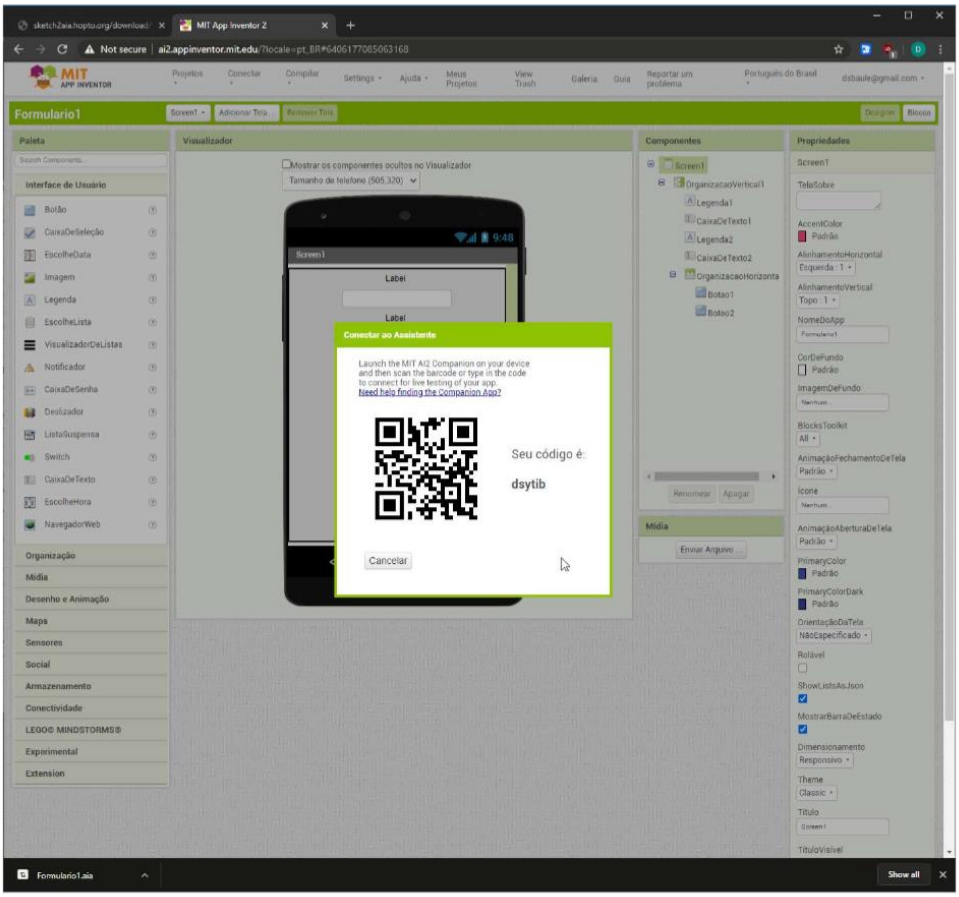
5 - Baixe o aplicativo MIT AI2 Companion em seu celular (Opcional)

O aplicativo MIT AI2 Companion permite testar o projeto em seu celular, pode ser baixado aqui:
https://play.google.com/store/apps/details?id=edu.mit.appinventor.aicompanion3&hl=pt_BR

6 - Clique em "Conectar > Assistente AI" para conectar ao MIT AI2 Companion (Opcional)



7 - Leia o QR Code informado com o aplicativo (Opcional)



Pronto, seu projeto deve aparecer na tela de seu celular

16. Avalie também quanto ficou correspondendo o wireframe gerado para o seu sketch *

Marcar apenas uma oval.

- ◯ Corresponde totalmente
- ◯ Corresponde largamente
- ◯ Corresponde pouco
- ◯ Não corresponde

Avaliando a ferramenta

17. Você acha a ferramenta Sketch2aia útil no processo de desenvolvimento de apps? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

18. Explique sua resposta *

19. A performance (tempo de resposta) do sistema é satisfatória? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

20. Você observou algum erro (bug) em relação a funcionalidade da ferramenta? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

21. Caso tenha observado algum bug, descreva-o

22. Você achou fácil de usar o sistema? *

Marcar apenas uma oval.

- Sim
- Não

23. Satisfação usando o Sketch2aia *

Marcar apenas uma oval por linha.

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
Eu acho que gostaria de usar esse sistema com frequência					
Eu acho o sistema desnecessariamente complexo					
Eu achei o sistema fácil de usar					
Eu acho que precisaria de ajuda de uma pessoa com conhecimentos técnicos para usar o sistema					
Eu acho que as várias funções do sistema estão muito bem integradas					
Eu acho que o sistema apresenta muita inconsistência					
Eu imagino que as pessoas aprenderão como usar esse sistema rapidamente					
Eu achei o sistema atrapalhado de usar					
Eu me senti confiante ao usar o sistema					
Eu precisei aprender várias coisas novas antes de conseguir usar o sistema					

Comentários gerais

24. O que mais você gostou da ferramenta Sketch2aia? *

25. Alguma sugestão de melhoria referente a ferramenta Sketch2aia? *

26. Mais algum comentário? *

Muito obrigado pela sua participação!

Este conteúdo não foi criado nem aprovado pelo Google.

Google Formulários

APÊNDICE D

Utilizando *Deep Learning* para Apoiar o Ensino de Design de Interface de Usuário na Educação Básica

Daniel de Souza Baulé
Dep. de Informática e Estatística
Universidade Federal de Santa Catarina
Florianópolis SC Brasil
daniel.baule@grad.ufsc.br

Christiane Gresse von Wangenheim
Dep. de Informática e Estatística
Universidade Federal de Santa Catarina
Florianópolis SC Brasil
c.wangenheimn@ufsc.br

Aldo von Wangenheim
Dep. de Informática e Estatística
Universidade Federal de Santa Catarina
Florianópolis SC Brasil
aldo.vw@ufsc.br

Jean C. R. Hauck
Dep. de Informática e Estatística
Universidade Federal de Santa Catarina
Florianópolis SC Brasil
jean.hauck@ufsc.br

Edson C. Vargas Júnior
Dep. de Matemática
Universidade Federal de Santa Catarina
Florianópolis SC Brasil
edson.junior@ufsc.br

Abstract. *Converting a user interface sketch into an App Inventor Wireframe has been a common need in computer education and in application development by end-users. Since this part of the process of mobile application development is lengthy and challenging, we present an approach to automate this process by prototyping user interfaces. Our approach Sketch2aia utilizes deep learning to detect user interface components and their position in a hand-drawn sketches, creating an intermediate representation of the user interface, and automatically generating App Inventor code for the respective wireframe. Our approach obtained average precision for classifying user interface components of 87,72% and the results of a preliminary evaluation with users show that it generates wireframes that resemble the sketches in terms of visual similarity. This approach is available as a Web tool and can be used to aid education in user interface design, as well as the development of mobile applications by end-users, effectively and efficiently.*

Resumo. *Transformar um sketch de uma interface de usuário em um wireframe usando o App Inventor tem sido uma necessidade comum tanto no ensino de computação quanto no desenvolvimento de aplicativos por usuários finais. Como essa parte do desenvolvimento de aplicativos móveis é desafiadora e demorada, apresentamos uma abordagem que automatiza esse processo por meio da prototipagem das interfaces de usuário. Nossa abordagem Sketch2aia emprega deep learning para detectar os componentes de interface do usuário e sua posição em um sketch desenhado à mão, criando uma representação intermediária da interface de usuário e gera automaticamente o código App Inventor do respectivo wireframe. A abordagem atinge uma precisão média de classificação dos componentes da interface de usuário de 87,72% e resultados de uma avaliação preliminar com usuários indicam que ela gera wireframes que se assemelham aos sketches em termos de similaridade visual. A abordagem está disponível como uma ferramenta Web e pode ser usada para apoiar de maneira eficaz e eficiente o ensino do design da interface de usuário, bem como o desenvolvimento de aplicativos móveis por usuários finais.*

CCS CONCEPTS

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

PALAVRAS CHAVES

Aplicativo móvel, Design de interface de usuário, Geração de código, *Deep learning*, Ensino, Desenvolvimento por usuários finais

1 Introdução

Existem hoje milhões de aplicativos para celular, desde redes sociais e entretenimento digital até soluções médicas. Assim, para um aplicativo obter sucesso nesse cenário atual, precisa-se mostrar uma experiência efetiva, eficiente e agradável, com uma interface de usuário (*User Interface - UI*) atraente. Consequentemente, o design de UI não é apenas um pequeno aspecto do desenvolvimento de um aplicativo, mas uma parte essencial para a estratégia de produto, enfatizando sua importância no processo de desenvolvimento de aplicativos para dispositivos móveis [41].

As competências relacionadas ao design da UI são importantes não apenas para os profissionais de Tecnologia da Informação, mas para qualquer pessoa, pois estão relacionadas às habilidades do século XXI [22]. E, embora as competências de engenharia de software sejam geralmente ensinadas apenas no ensino superior, as diretrizes curriculares, como o *K-12 Computer Science Framework* [12], indicam sua importância também na Educação Básica. Atualmente, existem vários esforços em diferentes países no ensino de engenharia de software, incluindo conceitos básicos de design de UI já no ensino fundamental e médio [30][14]. Também existem cursos que têm como objetivo o ensino de competências de design de UI incorporadas no contexto do ensino de computação por meio do desenvolvimento de aplicativos móveis usando o App Inventor [23], um ambiente de programação baseado em blocos para iniciantes. Parte desses cursos que adotam ação computacional [38] desafiam os alunos a desenvolver seu próprio aplicativo móvel, a fim de resolver um problema em sua comunidade. Incorporando o ensino dos conceitos de design de UI no desenvolvimento de software, seguindo um processo ágil, baseado no *design thinking* (Figura 1) [17][36][10], os alunos aprendem passo a passo como criar um aplicativo móvel para problemas complexos do mundo real [Ferreira et al., 2019a]. Esse também é o caso no desenvolvimento por usuários finais, sendo uma tendência importante, à medida que mais e mais aplicativos móveis são criados não por desenvolvedores profissionais de software, mas por pessoas com experiência em outros domínios [4][29].

Idealmente, em um curso que incorpora o ensino de competências de design de UI, os alunos identificam problemas e analisam o contexto, adotando uma abordagem de *design thinking*. Em seguida, eles começam a criar um protótipo do aplicativo fazendo *sketches*, que são representações simples desenhadas à mão. Eles são um meio visual eficaz para transmitir e discutir idéias e comparar diferentes alternativas de maneira simples, rápida e barata [18]. A partir dos sketches são criados os *wireframes* com o App Inventor, representando o *layout* e a estrutura da interface [34]. Depois que o *wireframe* é criado e revisado, ele é aprimorado definindo o design visual (cores, tipografia, imagem etc.), até se tornar um protótipo de alta fidelidade [34]. Quando o design é finalizado, as funções são implementadas, resultando no aplicativo funcional que é testado e compartilhado.

Como um processo iterativo de prototipagem, essas etapas podem ser repetidas várias vezes durante o design da UI, exigindo retrabalho sempre que alterações são feitas. Nesse contexto, especificamente, o processo para a geração de código de protótipos de UI tem um alto potencial de automação, pois é uma tarefa mecânica, demorada e propensa a erros [25][37].

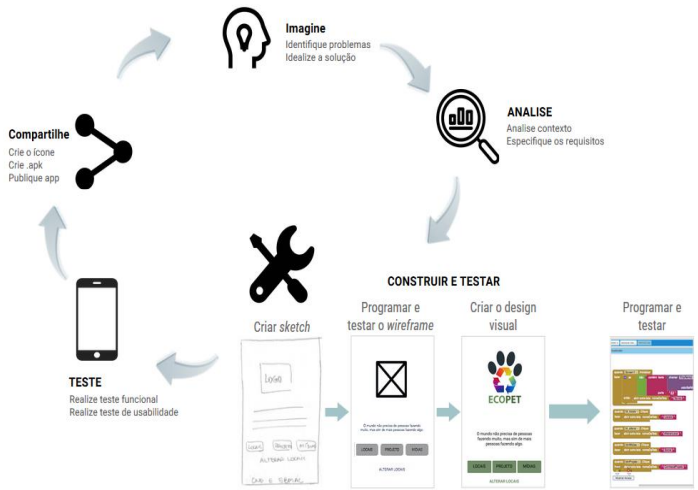


Figura 1: Processo ágil educacional para desenvolvimento de aplicativos baseado em *design-thinking* [13]

Embora os designers tipicamente utilizassem editores gráficos, como Photoshop ou Illustrator para projetar UI, uma grande variedade de ferramentas de design (como Sketch, Figma, Marvel ou Adobe XD) evoluíram para se tornarem ferramentas tudo-em-um, do design, prototipagem ao teste, mas ainda assim exigindo a codificação manual do design da UI. Por outro lado, IDEs modernos como Eclipse, Visual Studio ou Android Studio, possuem poderosos construtores interativos baseados em arrastar e soltar componentes para criação do código da UI. Mas, mesmo com as ferramentas atualmente disponíveis, a transição de *sketches* para código ainda consiste em recriar manualmente as UIs [7].

Nesse sentido, estão sendo criadas soluções para a geração automática de código de UI de web sites e aplicativos móveis, geralmente adotando abordagens de aprendizado de máquina [6]. Essas ferramentas convertem automaticamente *sketches* ou *wireframes* desenhados à mão em código de *front-end* ou representações intermediárias de código. Essa automação facilita o processo de desenvolvimento da UI, economizando tempo e esforço, além de evitar erros acidentais [27]. E, embora já existam várias abordagens que geram automaticamente o código de aplicativos móveis com base em *sketches* desenhados à mão, até o momento não existe uma solução que crie código para App Inventor [6].

Neste artigo apresentamos o Sketch2aia, uma solução para criar automaticamente código de *wireframes* de App Inventor com base em *sketches* desenhados à mão das UIs de um aplicativo. Para detectar automaticamente os componentes da UI nos *sketches,* adotamos uma abordagem de *deep learning*, criando uma representação intermediária da UI utilizada para criar automaticamente o código do App Inventor. A ferramenta está disponível online e pode ser usada para apoiar o ensino do design de UI como parte do ensino de computação, bem como no desenvolvimento por usuários finais, reduzindo o tempo gasto na criação manual de *wireframes* e, ao mesmo tempo, mantendo o processo criativo de desenho.

2 Trabalhos relacionados

Considerando a importância da estética e da usabilidade das UIs e, portanto, a importância do design de interface e a possível redução do esforço pela geração automática do código da UI, atualmente existe um pequeno número de esforços de pesquisa com esse objetivo [6]. Por outro lado, a relevância do tópico em si é apontada pela disponibilidade de soluções comerciais, como o Sketch2Code da Microsoft AI.Lab, entre outras.

Existem soluções para interfaces web e aplicativos móveis, com maior foco em Android. No entanto, nenhuma solução é voltada a criação de aplicativos com App Inventor e/ou ensino de engenharia de software/UI design. A maioria das abordagens é baseada em *sketches* como entrada, reconhecendo a importância do processo criativo de esboçar numa primeira etapa na criação de protótipos da UI, em contraste com abordagens que visam a automação completa do processo de design de UI. No entanto, uma deficiência de vários modelos existentes é o seu caráter mais exploratório, considerando apenas um número muito pequeno de componentes da interface de usuário. Por exemplo, Aşıroğlu et al. [3] e Robinson [34] consideram apenas quatro tipos diferentes de componentes (por exemplo, título, imagem, botão, entrada e parágrafo) e Ge [15] apenas sete, limitando sua aplicabilidade na prática.

A pesquisa com imagens de UI é ainda mais complicada devido à indisponibilidade de grandes conjuntos de dados, como, por exemplo, o ImageNet [32] para imagens em geral. Assim, é necessário um esforço considerável na criação de conjuntos de dados específicos, já que até conjuntos de dados sobre apps Android não fornecem sketches. Assim, vários estudos precisaram desenvolver seus conjuntos de dados [19][21][40].

Adotando diversas abordagens, outros estudos capturaram *screenshots* da UI por meio de *crawling* em sites e/ou lojas de aplicativos on-line. Em alguns casos, imagens ou *sketches* da UI foram gerados sinteticamente. Comparando-os com imagens e/ou *sketches* reais de UI, torna-se óbvio que existem diferenças significativas em relação à qualidade das imagens e à autenticidade, o que impacta no desempenho e na validade que não são discutidos na maioria dos casos. Uma exceção é Robinson [34], que relata uma redução do desempenho da abordagem de *deep learning* quando aplicada a *sketches* reais, enfatizando, assim, a importância de uma grande variedade de *sketches* para permitir que a abordagem de *deep learning* generalize melhor os estilos não conhecidos de *sketches*.

Em termos de técnicas de *Machine Learning* (ML), encontra-se uma grande variedade, de abordagens clássicas de ML com redes neurais convolucionais (CNN) recentes e diversas combinações. No entanto, a maioria das pesquisas adotou a CNN. Para poder detectar informações relacionadas ao contexto, como *widgets* de UI aninhados, algumas abordagens empregaram técnicas recorrentes de redes neurais convolucionais baseadas no campo de processamento de sinais e de linguagem natural, incluindo Unidades Recorrentes Fechadas, LSTM e Bi-LSTM [3][8][11][16][40].

Em termos de avaliação, a grande variedade de medidas utilizadas indica a falta de um padrão claro para a avaliação deste tipo de pesquisa. Além disso, como a maioria foca (em alguns casos exclusivamente) na análise de precisão, a avaliação não está necessariamente alinhada com as medidas propostas para a detecção de objetos (como a *mean Average Precision* (mAP), que foi usada apenas em uma pesquisa [37]). Essa falta de um padrão para avaliação também dificulta a comparação entre abordagens, bem como trabalhos futuros. Além disso, alguns estudos também avaliam a similaridade do código gerado adotando medidas de análise de texto, indicando ainda mais a falta de medidas específicas para esse tipo de pesquisa. Muito poucos também analisam a aplicabilidade das abordagens propostas por meio de estudos com usuários [7][11][34][37].

3 Metodologia de Pesquisa

Analisando os resultados de trabalhos relacionados a diferentes tipos de UI, desenvolvemos uma ferramenta Web para a geração automática de código de *wireframes* de App Inventor com base em *sketches,* seguindo uma abordagem multi-métodos.

Com base nos resultados de uma revisão sistemática da literatura [6], desenvolvemos um modelo de *deep learning* para a detecção de componentes da UI em *sketches*, seguindo o processo proposto por Amershi et al. [2]:

Análise de requisitos. Durante esta fase, o objetivo principal do modelo e suas características alvo são especificados seguindo Mitchell [24]. Isso também inclui a caracterização das entradas e saídas esperadas, especificando o problema.

Preparação de dados. Durante a coleta de dados, pesquisamos conjuntos de dados disponíveis e coletamos dados específicos capturando *screenshots* de UIs de projetos App Inventor e desenhando *sketches*. Isso inclui a seleção de conjuntos de dados genéricos disponíveis para pré-treinar um modelo (por exemplo, ImageNet para tarefas de classificação de imagens), bem como conjuntos de dados especializados para transferência de aprendizado para treinar o modelo específico. Os dados são preparados validando, limpando e condicionando os dados. Adotando o aprendizado supervisionado, *labels* que identificam os componentes da UI nos *sketches* foram atribuídos manualmente usando a ferramenta labelme. O conjunto de dados é dividido em um conjunto de treinamento para treinar o modelo e um conjunto de testes para executar uma avaliação imparcial do desempenho do modelo escolhido em dados não vistos [33].

Aprendizagem do modelo. Durante a fase de aprendizagem do modelo, foi escolhido um *framework* de *deep learning* apropriado e um modelo de aprendizado de máquina eficazes para um problema ou domínio comparável, além do volume e da estrutura dos dados. Adotando uma abordagem de *transfer learning* seguido de *fine tuning*, adotamos uma rede pré-treinada em um conjunto de dados genérico (ImageNet).

Avaliação do modelo. Foram definidas as métricas usadas para medir o sucesso alinhada com as metas a serem alcançadas e o tipo de problema enfocado pelo modelo de acordo com as medidas de desempenho comumente usadas para a detecção de objetos [44]. Em seguida, o(s) modelo(s) foram testados em relação a dados não vistos anteriormente com o conjunto de teste.

Desenvolvimento do modelo de geração de código .aia. Com base no modelo para detecção de componentes de UI em *sketches*, foi desenvolvido um módulo de software para geração automática do código .aia, responsável por arranjar os componentes em layouts compatíveis com o App Inventor e gerar o arquivo .aia correspondente [26], para a fácil importação no App Inventor.

Implantação como um sistema web. Adotando uma abordagem iterativa e incremental de desenvolvimento de software [20] foi desenvolvido um sistema web para o deployment da abordagem desenvolvida. Com base na análise de requisitos, foram definidos os casos de uso da ferramenta de software juntamente com o design de UI. Em seguida, foi implementado o módulo web, responsável pela interação do usuário e pela camada de persistência. Após a conclusão da implementação inicial do sistema, foram realizados testes de integração e, juntamente com o *feedback* do usuário, as correções e melhorias necessárias foram implementadas e testadas.

Avaliação preliminar. Para avaliar a qualidade da abordagem, realizamos uma avaliação preliminar usando a abordagem Goal/Question/Metric (GQM) [5] para definir o objetivo da avaliação e decompô-lo sistematicamente em características e subcaracterísticas. De acordo com as características definidas, coletamos dados por meio de um estudo com usuários em relação à similaridade visual percebida do *wireframe* gerado com o *sketch*. Os dados coletados foram analisados por estatística descritiva. Os resultados foram interpretados levando-se em consideração também outras observações feitas durante o estudo.

4 Sketch2aia

O objetivo desta abordagem é a geração automática de *wireframes* em App Inventor a partir de *sketches*. Isto inclui numa primeira etapa a detecção de componentes de UI em *sketches*, resultando em uma representação intermediária da UI identificando os componentes de UI e o *layout* e, a partir disso, a geração automática de código de App Inventor (Figura 2).

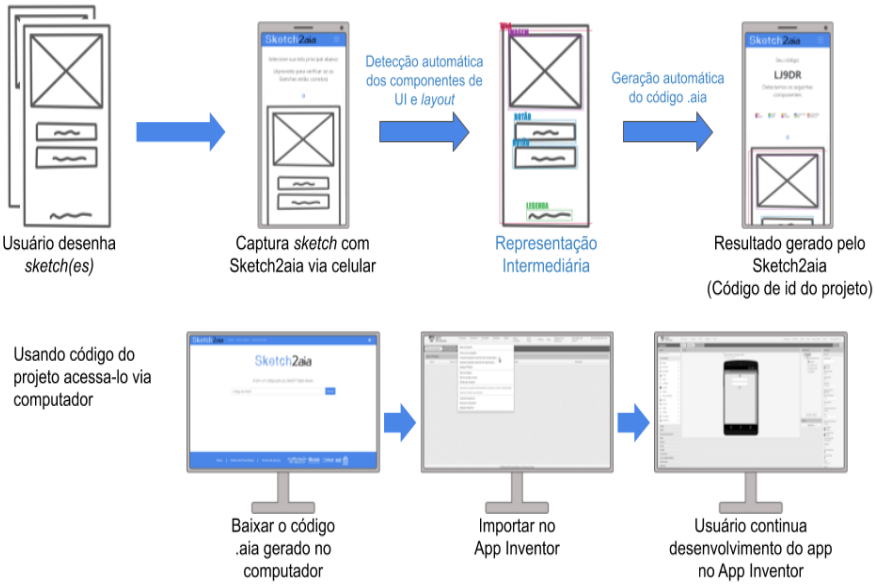


Figura 2: *Workflow* do Sketch2aia

4.1 Detecção de Componentes de UI em *Sketches*

Nesta etapa o objetivo é desenvolver um programa de computador que aprende com uma experiência (um conjunto de *sketches* de apps) em relação a tarefa de detectar componentes de UI e as suas posições em *sketches* e medidas (precisão, IoU, F1, mAP), medindo se seu desempenho na tarefa melhora de acordo com a experiência.

Devido ao grande número de tipos de componentes de UI, e a falta de um padrão de *sketch* para grande parte destes, foram selecionados os componentes que são utilizados com maior frequência em aplicativos de App Inventor [1]. Adicionalmente, foram considerados componentes mais recentes. Como resultado, são considerados nesse modelo os seguintes componentes: *Label, Button, Image, TextBox, CheckBox, ListPicker, Slider, Switch, Map.*

Preparação dos Dados. Criamos um novo conjunto de dados com *sketches* desenhados à mão da UI de aplicativos App Inventor com identificação dos componentes da UI e sua posição. Para isso, coletamos inicialmente 165 capturas de telas de UI de aplicativos da galeria App Inventor [http://appinventor.mit.edu] e de aplicativos desenvolvidos pela iniciativa Computação na Escola. As telas foram capturadas semi-automaticamente em *smartphones*. Selecionamos apenas telas que não contêm anúncios e que foram consideradas aceitáveis em relação a questões éticas. Com base nessas capturas de tela, 184 *sketches* foram desenhados por 28 voluntários de diversos campos, níveis de formação e faixas etárias, seguindo instruções básicas. Como resultado foram criados 279 *sketches* realistas de UIs de aplicativos App Inventor.

Os *sketches* foram rotulados manualmente com respeito aos componentes de UI e sua posição utilizando a ferramenta labelme, resultando em uma representação intermediária do design da UI (Figure 3).

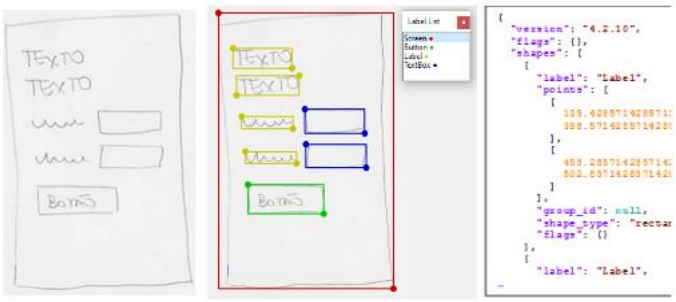


Figura 3: Rotulação dos *sketches*

Analisando a frequência de ocorrência dos componentes no conjunto de dados, observa-se que há uma grande diferença na frequência dos diferentes tipos de componentes de interface, com Botões (*Button*) e Legendas (*Labels)* compondo 69% dos componentes do conjunto de dados, e por outro lado mapas e sliders aparecendo em somente 1% dos UI, refletindo as diferenças no uso desses componentes em apps.

Dividimos o *dataset* em um conjunto de treinamento com 237 imagens (~85%) usado para o aprendizado e um conjunto de teste com 42 imagens (~15%). O *dataset* criado está disponível online: https://www.gqs.ufsc.br/sketch2aia-mobile-user-interface-sketches/.

Aprendizado do Modelo. Foi utilizado o *Darknet* [31], um *framework open-source* para redes neurais para implantação do modelo YoloV3 [32] visando a detecção de objetos em imagens. Como o conjunto de dados utilizado é relativamente pequeno, com apenas 279 imagens, há um risco considerável de *overfitting* se realizado o treinamento apenas com as imagens do conjunto. Para solucionar esse problema, exploramos a transferência de aprendizado a partir de uma rede YOLO pré-treinada com o conjunto de dados ImageNet [32], um conjunto de dados com mais de 1 milhão de imagens com objetos manualmente rotulados em 1000 categorias [35]. Esta estratégia permite utilizar um conjunto de dados pequenos para treinar uma rede efetivamente [43].

O *fine-tuning* da rede foi realizado de acordo com as orientações de Bochkovskiy [9], com alteração das camadas convolucionais e YOLO para se adequar ao número de classes de objetos considerados (9 tipos de componentes de UI, além da tela como um todo). As camadas YOLO foram configuradas para 10 classes, enquanto as camadas convolucionais imediatamente anteriores a elas foram configuradas com 45 filtros, seguindo a fórmula de Bochkovskiy [9]:

$$filters = (classes + 5) \times 3 = (10 + 5) \times 3 = 45 \quad (1)$$

Seguindo Bochkovskiy [9], o treinamento foi subdividido em *batches* de 64 imagens com 16 subdivisões, e o treinamento foi realizado até 20.000 *batches*:

$$max\ batches = classes \times 2000 = 10 \times 2000 = 20000 \quad (2)$$

Foram realizados *backups* dos pesos da rede a cada 1000 *batches*, para posterior avaliação da evolução da rede.

Avaliação de Desempenho. A qualidade de um modelo de detecção de objeto é tipicamente baseada no IoU (*Intersection Over Union*), uma medida baseada no índice Jaccard que avalia a sobreposição entre duas caixas delimitadoras, a caixa delimitadora verdade e a caixa delimitadora prevista [44]. Outra métrica frequentemente usada para detecção de objetos é a Precisão Média (*Average Precision* - AP) [21]. O AP é definido como a precisão de detecção média em diferentes *recalls* e geralmente é avaliado de uma maneira específica da classe de objeto. Para comparar as classes gerais de objetos de desempenho, o AP médio (mAP) calculado sobre todas as classes de objetos é comumente usado como a métrica final de desempenho. Além do IoU e mAP, o *F1 score* também é uma métrica utilizada para a avaliação deste tipo de modelo. O F1 score é a média harmônica da precisão e *recall*, transmitindo informações sobre o equilíbrio entre estes [Robinson 2018]. Um *F1 score* alto indica que o sistema executa uma detecção que é exata e completa [34].
Durante a criação do modelo foram analisadas todas essas métricas permitindo também a comparação com modelos existentes. A Figura 4 mostra a evolução do desempenho do modelo Sketch2aia durante o treinamento de acordo com estas métricas.

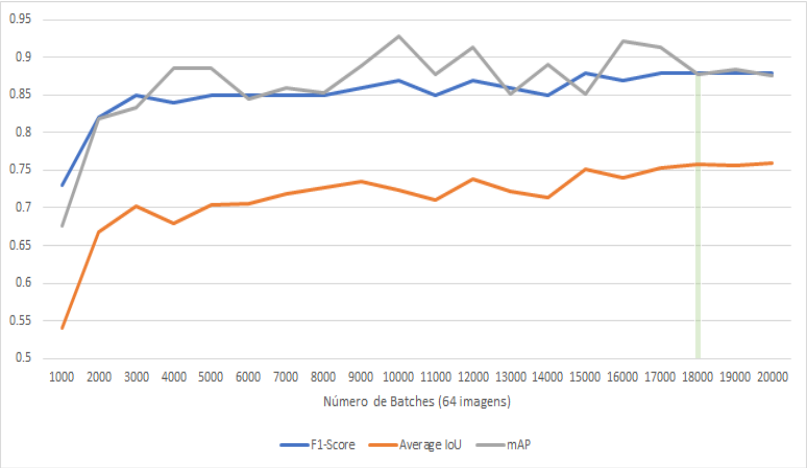


Figura 4: Evolução do desempenho durante o treinamento

O melhor resultado obtido foi após 18.000 *batches*, apresentando o melhor F1 *score* de 0.88 com valores mais balanceados de IoU médio e mAP, quando comparado aos demais resultados (Figura 5).

Resultado após 18000 Batches					
F1-Score	0.88	AP Screen	97.62%	AP Image	100.00%
Average IoU	75.85%	AP Label	80.88%	AP Switch	100.00%
mAP	87.72%	AP Button	97.10%	AP Slider	100.00%
		AP TextBox	81.44%	AP Map	100.00%
		AP CheckBox	95.21%	AP ListPicker	25.00%

Figura 5: Melhores resultados obtidos (total e por componente)

A Figura 5 mostra a precisão média obtida para cada componente no conjunto de teste. A presença de componentes com 100% de precisão provavelmente se deve ao tamanho limitado do conjunto de teste e consequentemente baixa frequência desses componentes (por exemplo, somente 4 *ListPickers*), mas ainda indica um bom resultado. Outro possível fator para a baixa precisão obtida com alguns componentes é a ausência de características distintas quando comparados aos demais. *ListPicker*, *Button* e *TextBox*, por exemplo, são muito similares visualmente com somente pequenos detalhes determinando a diferença entre eles.

Comparando com outros modelos existentes, o F1 *score* médio obtido por nosso modelo (0.88) é maior que o melhor índice obtido por Robinson [34] (0.81, em imagens) e que o índice médio obtido por Huang et al. [18] (0.775). Quanto ao único modelo encontrado que também usou a métrica mAP [37][28] obtendo uma mAP de 84.9%, enquanto o modelo Sketch2aia obteve uma mAP de 87.72%. Dessa forma, o modelo Sketch2aia demonstra resultados de avaliação de desempenho equivalente ou melhor que os modelos existentes encontrados na revisão da literatura, mostrando um suporte a detecção de componentes de UI com precisão.
O modelo criado está disponível online https://www.gqs.ufsc.br/sketch2aia-mobile-user-interface-sketches/.

4.2 Gerando Código de *Wireframes*

A saída da detecção de componentes de UI em *sketches* é uma representação intermediária, na forma de uma lista de elementos detectados, seus respectivos níveis de confiança e suas posições na imagem (coordenadas do ponto central, largura e altura) conforme ilustrado na Figura 6.

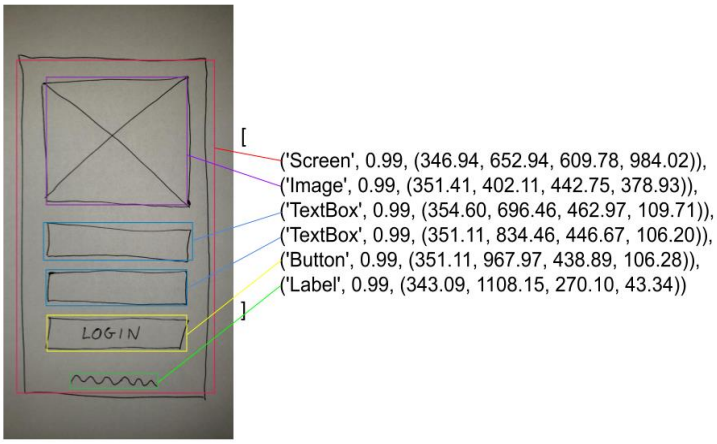


Figura 6: Exemplo de saída da detecção de componentes de UI em *sketches*

O primeiro passo para gerar o código de *wireframes* App Inventor é a eliminação de possíveis sobreposições de componentes, para isso checamos para cada par de elementos se a área de sobreposição deles é maior que 50% da área do menor elemento entre eles, eliminando o com menor índice de confiança em caso afirmativo.

Mesmo após a eliminação das sobreposições, os componentes não estão organizados em forma de *layouts*, mas por suas posições no espaço explicitamente definidas. Esta representação não é compatível com o App Inventor, que requer que os componentes estejam organizados em um dos seguintes *layouts* para obter uma organização mais complexa: OrganizaçãoHorizontal, *HorizontalScrollArrangement*, OrganizaçãoEmTabela, OrganizaçãoVertical ou *VerticalScrollArrangement*. Devido a sua simplicidade e comportamento mais previsível, utilizamos nessa implementação somente a organização horizontal e a organização vertical. Para organizar os componentes em *layouts* verticais e horizontais, é utilizado um algoritmo recursivo (Figura 7).

Com os componentes organizados em *layouts* compatíveis, é realizada para cada *sketch* uma tradução direta para um arquivo .scm, que define o *wireframe* da tela correspondente dentro do arquivo .aia (Figura 8). O arquivo .scm do App Inventor encapsula uma estrutura JSON que contém todos os componentes visuais utilizados no aplicativo [26], preenchido com valores determinados empiricamente, por meio da análise de diversos projetos de App Inventor (Figura 8). Para cada tela é gerado também um arquivo .bky vazio. O arquivo .bky encapsula uma estrutura XML com todos os blocos de programação usados na lógica do aplicativo [26].

```
função alinhar(componentes, orientação)
inicio
  //Ordenar os componentes da lista de acordo com a
    orientação do alinhamento atual
  componentes = ordenarLista(componentes, orientação)
  //Enquanto a lista de componentes não estiver vazia
  para cada componenteAtual em componentes
  Inicio
       componentes.remove(componenteAtual)
   //Verifica se existem na lista componentes alinhados
   com o atual no sentido contrário
   [...]
   //Para cada componente alinhado novo, verifique também
   se existem outros
   // componentes alinhados com este
      para cada componenteAnalisado em componentesAAnalisar
   Inicio
         [...]
      Fim_para
   //Se não estiver alinhado com outros componentes,
   adicione na lista,
   // senão alinhe estes primeiro
   se novosComponentes.vazia() então
        alinhamentoAtual.adiciona(componenteAtual)
   senão
                [...]
     alinhamentoAtual.adiciona (alinhar
    (novosComponentes, ~orientação))
   fim_se
  fim_para
  retorna tupla(orientação, alinhamentoAtual)
fim
```

Figura 7: Algoritmo para agrupamento dos elementos detectados em *layouts* compatíveis com o App Inventor

Após a geração dos códigos de *wireframe* correspondentes a todas as telas, o arquivo de propriedades é gerado conforme as configurações do projeto (Nome, Tela Principal, etc.).


```
{
    "authURL": [
        "ai2.appinventor.mit.edu"
    ],
    "YaVersion": "206",
    "Source": "Form",
    "Properties": {
        "$Name": "Screen1",
        "$Type": "Form",
        "$Version": "27",
        "AppName": "Test",
        "Title": "Screen1",
        "Uuid": "0",
        "$Components": [
            {
                "$Name": "OrganizacaoVertical1",
                "$Type": "VerticalArrangement",
                "$Version": "3",
                "AlignHorizontal": "3",
                "AlignVertical": "1",
                "Height": "-2",
                "Width": "-2",
                "Uuid": "6204202716",
                "$Components": [
                    {
                        "$Name": "ListaSuspensa1",
                        "$Type": "Spinner",
                        "$Version": "1",
                        "AlignHorizontal": "1",
                        "AlignVertical": "1",
                        "Height": "-1",
                        "Width": "-1",
                        "Uuid": "7956601802"
                    },
```


Figura 8: Exemplo de código de *wireframe* para App Inventor

Os arquivos gerados são compactados em um arquivo .zip seguindo a estrutura de arquivos esperada pelo App Inventor, salvo como um arquivo .aia que pode ser importado diretamente no App Inventor.

4.3 Ferramenta Web

Como resultado, a ferramenta Sketch2aia foi implementada como uma aplicação web. A ferramenta permite que o usuário faça o *upload* de até 6 imagens de *sketches* de UI de aplicativos móveis (Figura 9). O Sketch2aia então detecta automaticamente os componentes de UI e gera o código de App Inventor (um arquivo .aia) correspondente. O arquivo pode ser então importado pelo usuário no ambiente App Inventor, onde ele(a) pode continuar o desenvolvimento do aplicativo.

Figura 9: *Upload* de *sketches* para a ferramenta Sketch2aia

A ferramenta também permite que o usuário configure algumas opções, como o nome do projeto, a tela principal, e a escolha entre a utilização de EscolheLista ou ListaSuspensa, dois componentes com função semelhante no App Inventor, mas esteticamente diferentes.

Além disso, é possível visualisar uma *preview* da detecção realizada, possibilitando a identificação de diferenças entre os componentes esperados e os realmente detectados antes da sua importação para o App Inventor. O código gerado pela ferramenta também possibilita o fácil compartilhamento de projetos e seu *download* usando um computador, mesmo se o upload das imagens dos sketches foi realizado via celular (Figura 10).

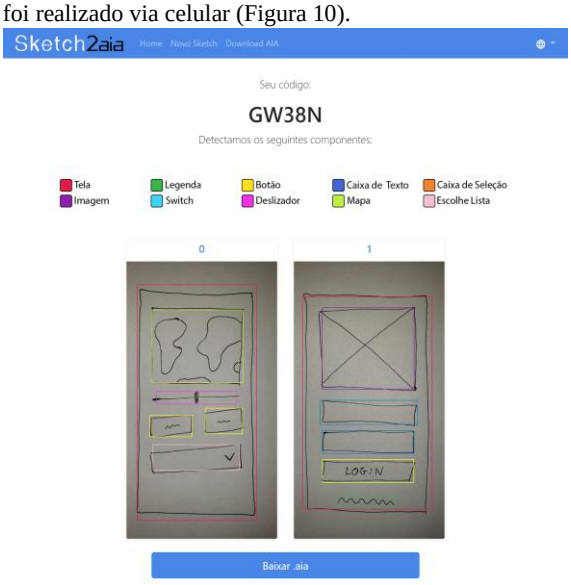


Figura 10: Resultado fornecido pela ferramenta Sketch2aia

A ferramenta web está disponível online: http://apps.computacaonaescola.ufsc.br:4555/

5 Avaliação Preliminar

5.1 Definição da Avaliação

O objetivo consiste em avaliar se os *wireframes* gerados correspondem ao design da UI do *sketch*. Portanto, utilizamos um estudo de usuário nesta avaliação preliminar. Neste estudo são apresentados aos participantes 10 pares de *sketches* e capturas de tela dos *wireframes* gerados com Sketch2aia. Esses foram selecionados aleatoriamente a partir do conjunto de teste/validação. Os participantes do estudo também são solicitados a desenhar um *sketch* de um aplicativo móvel e, em seguida, usar a ferramenta web Sketch2aia com uma foto deste *sketch* gerando o *wireframe.* Na sequência, solicita-se que eles baixem o código .aia criado e o importem no ambiente do App Inventor, executando a interface do usuário do aplicativo em seus *smartphones*. Referente a esses 11 pares, os participantes classificaram a correspondência de cada um dos *wireframes* de UI com os *sketches* em uma escala ordinal de 4 pontos, variando de não corresponde a corresponde totalmente.

5.2 Execução da Avaliação

O estudo com usuários ocorreu no contexto da iniciativa Computação na Escola/INCoD/INE/UFSC em Florianópolis/SC em junho de 2020. A população do estudo foi composta por 22 alunos e professores com formação em ciência da computação e/ou design, bem como 6 jovens e 1 professor representando usuários-alvo de um contexto da educação básica. O estudo foi realizado on-line, fornecendo instruções detalhadas aos usuários, junto com um questionário. A participação foi voluntária, com taxa de resposta de 75.68%.

5.3 Resultados

Com base nas respostas dos participantes, podemos observar que de forma geral os *wireframes* gerados correspondem largamente ou totalmente aos 10 *sketches* nos 10 exemplos fornecidos, com resultados similares sendo observados com os *sketches* desenhados pelos próprios participantes, como mostra a Figura 11.

Uma exceção é o protótipo 4, que corresponde pouco segundo a maioria dos voluntários (Figura 12). Esta avaliação provavelmente se deve à detecção incorreta de *CheckBox* na parte superior da imagem, devido ao estilo de desenho utilizado para *Label* no *sketch*, com a combinação de letras e *placeholders.*

Também foram obtidos resultados positivos quanto a avaliações do processo de uso da ferramenta. 100% dos participantes avaliaram que a ferramenta Sketch2aia é útil no processo de desenvolvimento de apps e 96.6% avaliaram que a ferramenta é fácil de usar.

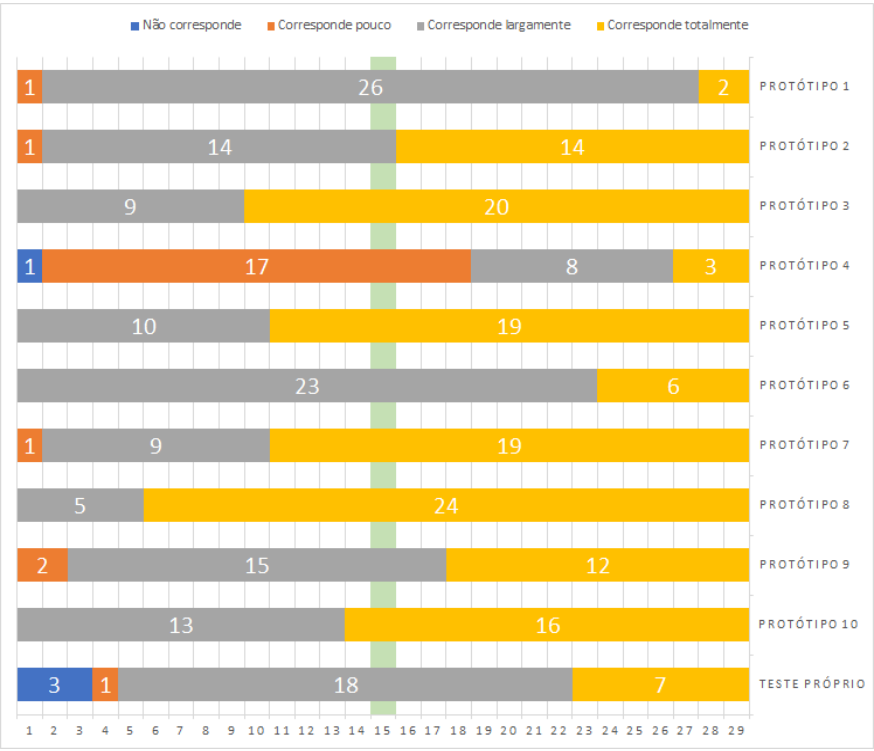


Figura 11: Resultados da avaliação preliminar

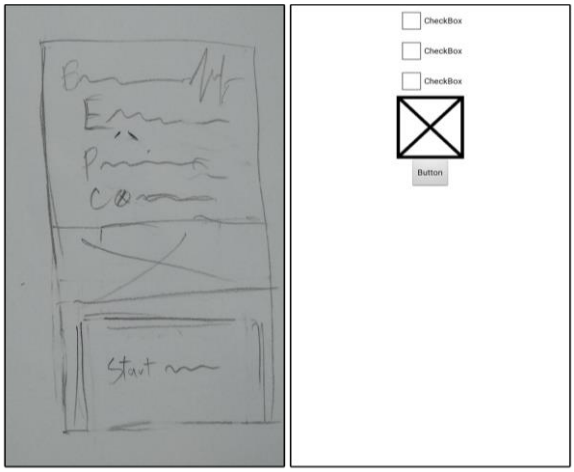


Figura 12: *Sketch* e *wireframe* gerado do Protótipo 4

Dentre os principais pontos positivos da ferramenta Sketch2aia destacados por usuários estão sua praticidade, facilidade de uso e economia de tempo no processo de desenvolvimento de aplicativos em App Inventor. Quanto a possíveis oportunidades de melhoria, os voluntários destacaram os layouts dos componentes no *wireframe* de App Inventor gerado pela ferramenta, algo dificultado pelas características internas do próprio App Inventor.

6 Discussão

De forma geral a abordagem se demonstrou precisa na detecção de objetos quanto à geração de *wireframes* em relação a similaridade visual com os *sketches*.

Considerando que a avaliação de performance da detecção de objetos indicou um F1 *score* de 0.88, pode-se concluir que a abordagem geralmente identifica os componentes de UI em *sketches* corretamente. Com a exceção de *ListPicker*, todos os tipos de componentes obtiveram AP acima de 80%. A performance abaixo da média obtida com *ListPicker* se dá provavelmente devido a sua baixa frequência de aparição no conjunto de dados e sua similaridade com outro componentes mais frequentes, como *Button* e *TextBox*. Futuramente, a introdução de mais exemplos de *ListPicker* no conjunto de dados, ou até mesmo a criação de um padrão diferente com características mais distintas para sua representação em *sketches* podem ajudar a solucionar estes problemas. Comparando esses resultados com o estado da arte também se evidencia a qualidade da abordagem por demonstrar valores de desempenho acima dos trabalhos relacionados.

Os resultados do estudo de usuário também indicam que os *wireframes* gerados pela ferramenta correspondem largamente aos *sketches* utilizados. A maior parte das questões indicadas se referem ao *layout* das UI geradas, como por exemplo, o alinhamento dos elementos de UI (Figura 13).

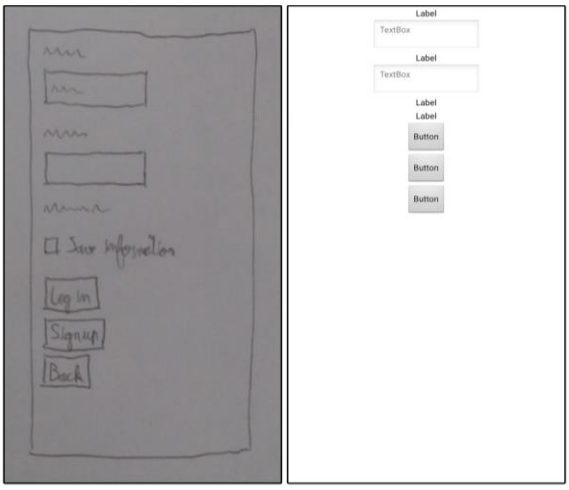


Figura 13: Exemplo com problema de alinhamento e espaçamento

Também foram observados problemas com espaçamento entre componentes, com a inclusão de espaços em branco sendo uma possível solução para melhor aproximação do *layout* do *sketch*.

Testes de performance também indicam que o sistema pode ser uma maneira eficiente de reduzir o tempo utilizado para design de UI, considerando que um desenvolvedor de UI/UX pode levar em média mais de 1 hora para criar um protótipo de alta fidelidade [7], enquanto a geração automática de código de *wireframe* baseada em *sketches* de nossa abordagem leva cerca de 13.55 segundos (para um aplicativo com três telas).

Comparado com outras abordagens, como, por exemplo, REDRAW [25], nossa abordagem é capaz de prototipar várias telas (no máximo 6) de uma aplicação ao mesmo tempo, eliminando a necessidade de prototipar individualmente cada tela e, em seguida, combinar manualmente em uma única aplicação. Sketch2aia também é capaz de detectar um conjunto de nove componentes de interface do usuário mais utilizados em aplicativos de App Inventor, superando conjuntos muito reduzidos, que limitam algumas abordagens relacionadas [3][34][15].

Em vez de criar código para ambientes de desenvolvimento móvel profissional, a geração de código do App Inventor também fornece um suporte inexistente até nesse momento para o desenvolvimento de aplicativos móveis pelo usuário final, bem como o contexto de ensino da computação usando o ambiente de programação baseado em blocos App Inventor, que permite a qualquer pessoa criar aplicativos móveis, projetando a interface com um simples desenho.

Ameaças à validade. Devido às características desse tipo de pesquisa, identificamos possíveis ameaças e aplicamos estratégias de mitigação para minimizar seu impacto. Para criar um conjunto de dados autêntico e variado, criamos *sketches* manualmente para as UIs de aplicativos App Inventor, envolvendo uma amostra de 28 participantes, reduzindo o risco de *sketches* que não representam adequadamente *sketches* realistas desenhados por um público-alvo com diversas origens e idades. O tamanho da amostra de 279 *sketches* de UI de aplicativos do App Inventor selecionados aleatoriamente na Galeria do App Inventor e de aplicativos desenvolvidos pela iniciativa Computação na Escola fornecem uma amostra representativa das UIs de aplicativos desenvolvidos com App Inventor.

A seleção do modelo de *deep learning* foi feita sistematicamente com base na literatura como resultado de testes informais, por exemplo, comparando diferentes versões do YOLO para maximizar os resultados de desempenho. A seleção das medidas de desempenho também foi realizada com base na literatura que indica as medidas comuns utilizadas para esse tipo de aplicação, a fim de assegurar uma avaliação adequada.

Com relação ao estudo de avaliação, definimos sistematicamente o estudo usando a abordagem GQM [5]. Em comparação com as avaliações em trabalhos relacionados, também consideramos aceitável a comparação de um total de 11 *sketches* com os *wireframes* gerados, para fornecer as primeiras idéias sobre a correspondência dos resultados. Com relação ao tamanho da amostra, nossa avaliação utilizou dados coletados de 29 participantes. No contexto de uma avaliação preliminar, esse é um tamanho de amostra aceitável que permite obter os primeiros resultados. No entanto, como os dados foram obtidos em apenas um contexto, serão necessários mais estudos para aumentar a validade externa.

7 Conclusão

Neste artigo, apresentamos o Sketch2aia, uma abordagem para criar protótipos de UI de aplicativos móveis para App Inventor automaticamente. Uma avaliação preliminar da abordagem demonstra que ela é capaz de detectar e classificar com precisão os componentes da interface do usuário e sua posição em um *sketch,* além de gerar códigos de *wireframes* App Inventor que são visualmente semelhantes aos *sketches*. Um primeiro *feedback* de usuários também fornece uma indicação de que a abordagem pode oferecer suporte efetivo e eficiente ao processo de design da interface de usuário como parte do desenvolvimento de aplicativos móveis. Estamos planejando como trabalhos futuros ampliar a avaliação em escala também no contexto educacional, bem como usá-la para aprimorar o suporte da ferramenta, incluindo a recomendação de diretrizes e exemplos de interfaces de usuário.

AGRADECIMENTOS

Gostaríamos de agradecer a todos os participantes por sua ajuda para desenhar *sketches* e avaliar a abordagem.

Este trabalho teve suporte do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico – www.cnpq.br), uma entidade do governo brasileiro com foco no desenvolvimento científico e tecnológico.

REFERÊNCIAS

N. d. C. Alves, C. Gresse von Wangenheim, J. C.R. Hauck. A Large-scale Analysis of App Inventor Projects. arXiv:2006.11327 [cs.CY], 2020.

S. Amershi et al. 2019. Software engineering for machine learning: a case study. In *Proc. of the 41st Int. Conference on Software Engineering: Software Engineering in Practice*, IEEE Press, 291-300.

B. Aşıroğlu et al., 2019. Automatic HTML Code Generation from Mock-up Images Using Machine Learning Techniques. In *Proc. of the Scientific Meeting on Electrical-Electronics and Biomedical Engineering and Computer Science*, Istanbul, Turkey, 1-4.

B. R. Barricelli, F. Cassano, D. Fogli, A. Piccinno. 2019. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software*, 149, 101-137.

V. R. Basili et al. 1994. Goal, Question Metric Paradigm. In J. J. Marciniak, *Encyclopedia of Software Engineering*, Wiley-Interscience, New York.

D. d. S. Baulé, C. Gresse von Wangenheim, A. von Wangenheim, J. C. R. Hauck. Recent Progress in Automated Code Generation from GUI Images using Machine Learning Techniques. Journal of Universal Computer Science, 26(9), 2020.

T. Beltramelli, 2019. Hack your design sprint: wireframes to prototype in under 5 minutes"; Medium, https://uxdesign.cc/hack-you-design-sprints-wireframes-to-prototype-in-under-5-minutes-b7b95c8b2aa2

T. Beltramelli, 2018. pix2code: Generating Code from a Graphical User Interface Screenshot. In *Proc. of the SIGCHI Symposium on Engineering Interactive Computing Systems*, ACM, New York, NY, USA, Article 3, 1–6.

A. Bochkovskiy, 2019. Yolo-v3 and Yolo-v2 for Windows and Linux, Github, https://github.com/AlexeyAB/darknet/

T. Brown, 2008. Design thinking. *Harvard Business Review*, 86(5), 84–92.

C. Chen et al. 2018. From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation. In *Proc. of the Int. Conference on Software Engineering*, ACM, New York, NY, USA, 665–676.

CSTA. 2017. *K-12 Computer Science Framework*, http://k12cs.org/

M. N. F. Ferreira et al. 2019a. Learning user interface design and the development of mobile applications in middle school. *ACM Interactions*, 26(4).

M. N. F. Ferreira et al. 2019b. Ensinando Design de Interface de Usuário na Educação Básica: Um Mapeamento Sistemático do Estado da Arte e Prática. In *Anais do Workshop de Informática na Escola*, Brasília, Brasil.

X. Ge, 2019. Android GUI search using hand-drawn sketches. In *Proc. of the 41st Int. Conference on Software Engineering*, IEEE Press, 141–143.

Y. Han et al. 2018. CSSSketch2Code: An Automatic Method to Generate Web Pages with CSS Style. In *Proc. of the 2nd Int. Conference on Advances in Artificial Intelligence*, Barcelona, Spain, 29–35.

R. Hartson, P. Pyla, 2012. *The UX Book: Process and Guidelines for Ensuring a Quality User Experience*, Morgan Kaufmann.

F. Huang et al. 2019. Swire: Sketch-based User Interface Retrieval. In *Proc. of the CHI Conference on Human Factors in Computing Systems*, ACM, New York, NY, USA, 1–10.

A. Kumar, 2018. Automated front-end development using deep learning, *Medium Insight.*

B. Larman, V. R. Basili, 2003. Iterative and incremental development, *IEEE Computer*, 36(6), 47-56.

Y. Liu et al. 2018. Improving pix2code based Bi-directional LSTM. In *Proc. of the IEEE Int. Conference on Automation, Electronics and Electrical Engineering*, Shenyang, China, 220-223.

R. Lozner, 2013. Where is Design in the K12 Curriculum? *AIGA*, https://www.aiga.org/where-is-design-in-the-k12-curriculum

MIT, 2020. *App Inventor*, http:// http://appinventor.mit.edu

T. M. Mitchell, 1997. *Machine Learning*. McGraw-Hill Education, New York.

K. Moran et al. 2018. Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps. *IEEE Transactions on Software Engineering*, 46(2), 196-221.

E. Mustafaraj et al. 2017. Identifying original projects in App Inventor. In *Proc. of the 30th Int. Flairs Conference*, Marco Island, USA.

Ozkaya, 2019. Are DevOps and Automation Our Next Silver Bullet? *IEEE Software*, 36(4), 3-95.

V. P. S. Pandian et al. 2020. Blu: What GUIs are made of. In *Proc. of the 25th Int. Conference on Intelligent User Interfaces*, ACM, New York, NY, USA, 81–82.

F. Paternò, End User Development: Survey of an Emerging Field for Empowering People. *ISRN Software Engineering*, vol. 2013, Article ID 532659.

F. Pinheiro et al., 2018. Teaching Software Engineering in K-12 Education: A Systematic Mapping Study. *Informatics in Education*, 17(2), 167–206.

Redmon, 2016; *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/.

Redmon, A. Farhadi, 2019. *YOLOv3: An Incremental Improvement*. arXiv preprint arXiv:1804.02767.

B. D. Ripley, 1996. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge.

A. Robinson, 2018. S*ketch2code: Generating a website from a paper mockup*. Dissertation, University of Bristol, UK.

O. Russakovsky et al., 2014. *ImageNet Large Scale Visual Recognition Challenge*. arXiv preprint arXiv:1409.0575.

T. Silva da Silva et al. 2011. User-Centered Design and Agile Methods: A Systematic Review. In *Proc. of the Agile Conference*, Salt Lake City, UT, USA, 77-86.

S. Suleri et al. 2019. Eve: A Sketch-based Software Prototyping Workbench. In *Proc. of the Conference on Human Factors in Computing Systems*, ACM, New York, NY, USA, 1–6.

Tissenbaum et al. 2019. From computational thinking to computational action. *Communications of the ACM,* 62(3), 34–36.

M. C., Valentim et al. 2017. The Students' Perspectives on Applying Design Thinking for the Design of Mobile Applications, In *Proc. of the 39th Int. Conference on Software Engineering*, IEEE Press, 77–86.

E. Wallner, 2018. Turning Design Mockups into Code with Deep Learning. Floydhub (2018) Article available at: https://blog.floydhub.com/turning-design-mockups-into-code-with-deep-learning

A. I. Wasserman, 2010. Software engineering issues for mobile application development. In *Proc. of the Workshop on Future of Software Engineering Research*. ACM, New York, NY, USA, 397–400.

K. Wada, 2016. *labelme: Image Polygonal Annotation with Python,* Github, https://github.com/wkentaro/labelme

J. Yosinski et al. 2014. How transferable are features in deep neural networks?. In *Proc. of the 27th Int. Conference on Neural Information Processing Systems*, Montreal, Canada, 3320-3328.

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