UNIVERSIDADE FEDERAL DE SANTA CATARINA - CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Desenvolvimento de Sketch2Penpot

Joaquim Boschini Belo

Florianópolis
2023

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Desenvolvimento de Sketch2Penpot

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: Joaquim Boschini Belo

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

Florianópolis
2023

Sumário

Lista de figuras

Lista de tabelas

Lista de abreviaturas

AP - Average Precision
CNN – Convolutional Neural Network
DL – Deep Learning
GUI – Graphical User Interface
IA – Inteligência Artificial
IEEE - Institute of Electrical and Electronics Engineers
IoU – Intersection Over Union
NI – Não Informada
UI – User Interface

Resumo

Ao longo dos últimos anos, aplicativos para dispositivos móveis tem cada vez mais se tornado parte fundamental do dia a dia da população. O aumento significativo em sua importância causou uma expansão no número de aplicativos no mercado, intensificando a competição, tornando o tempo de desenvolvimento um fator para o sucesso. Uma parte de grande importância é a interface de usuário. Normalmente o seu desenvolvimento envolve uma fase em que são criados *sketches* em caneta e papel, que depois são recriados em formato de wireframe em uma ferramenta de de *design* gráfico, no qual *designers* podem refinar e iterar o *design* do interface de usuário. Este processo é tipicamente feito manualmente e que consome um esforço considerável. Mesmo existindo diversos trabalhos que geram *wireframes* de aplicativos a partir de *sketches*, poucos focam em *wireframes* em aplicações de *design*, e nenhuma para Penpot. Assim, este trabalho visa criar a ferramenta Sketch2Penpot, que utilizando técnicas de *deep learning,* gere um projeto Penpot automaticamente a partir de sketches de interfaces de usuários.

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

1. Introdução

1.1 Contextualização

Com o contínuo aumento do uso aplicativos para dispositivos móveis, a necessidade de processos ágeis de desenvolvimento e iteração também aumenta (Barua, 2022). Uma das principais áreas na qual existe um grande investimento e retorno é no desenvolvimento das interfaces de usuário (Barua, 2022).

O desenvolvimento de interfaces normalmente envolve um processo de prototipação em que são elaboradas *sketches* que são representações simples dos elementos gráficos, tipicamente feitas em papel e canetas. A partir dos *sketches* são criados *wireframes*, estruturas de *design* sem detalhes visuais como por exemplo cores, que definem a estrutura base da interface (Abdelhamid, 2020). E ao final é feito o *design* visual adicionando os detalhes de cor, tipografia, ícones e imagens tornando os *wireframes* em uma representação mais fiel do produto.

O desenvolvimento de *wireframes* e do *design* visual é tipicamente feito usando ferramentas de *design* gráfico. Um exemplo é a Penpot (Kaleidos, 2023) é uma ferramenta de *design*, de código aberto que permite a prototipação de diversos produtos, dentre eles interfaces de usuário. Tem como objetivo aproximar desenvolvedores e *designers* de maneira a tornar o processo criativo mais escalável, sendo uma alternativa gratuita a outras ferramentas, como por exemplo Figma (Kaleidos, 2023). Penpot utiliza elementos simples, como linhas e círculos, que podem ser agrupados de forma a criar elementos mais complexos. Projetos podem ser importados e exportados por meio de arquivos .json e .svg ou no formato proprietário .penpot.

Pela natureza custosa e iterativa do processo, a automação se torna de grande valor (Moran et al., 2020). Uma das partes que pode ser automatizada é a geração de *wireframes* a partir de *sketches* e possibilitando assim uma continuação do processo de *design* adicionando elementos do *design* visual como cor e fontes. Nos últimos anos foram desenvolvidas várias soluções para a transformação de *sketches* para *wireframes* em HTML, *Sketch*, React, Android Studio entre outros (Baulé et al. 2021), utilizando diversas técnicas de visão computacional clássica (Huang et al., 2019), redes neurais convolucionais (Yun et al., 2018) ou alguma combinação destas técnicas (Robinson, 2019). Porém, atualmente ainda não existe uma ferramenta que permita a criação automática de *wireframes* no Penpot a partir de *sketches* de papel.

Visando a importância da competência de *design* de interface, este conteúdo também está começando a ser abordado no ensino de computação na Educação Básica. No contexto do ensino de computação por meio de desenvolvimento de aplicativos móveis

neste estágio escolar são usados tipicamente ambientes de programação baseados em blocos. Um exemplo é o App Inventor (MIT, 2020) é uma ferramenta baseada em blocos que permite o desenvolvimento de aplicativos para dispositivos móveis. Ele permite tanto o desenvolvimento funcional de app, quanto também o seu *design* de interface de usuário. Para o *design* de interface o App Inventor oferece diversos elementos de *design* de interface tais como botões, *switches*, *sliders*, seletores de data, entre outros, que também podem ser customizadas em relação às cores, fontes, tamanhos, etc. Assim, neste contexto o presente trabalho visa a aplicação da solução de criação automatizada de wireframes a serem posteriormente implementados no App Inventor, limitando assim o escopo dos elementos de UI a elementos possíveis de serem criados na biblioteca core do App Inventor.

1.2 Objetivos

Objetivo geral

O objetivo geral deste trabalho é desenvolver um modelo de geração automática de *wireframes* de apps App Inventor no Penpot a partir de *sketches* de papel. 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* em um arquivo que pode ser importado na ferramenta Penpot a partir do modelo semântico obtido.

Objetivos Específicos

O1. Analisar a fundamentação teórica sobre aprendizagem de elementos de *design* de interface de usuário em apps App Inventor, a representação de projetos na ferramenta Penpot, e *deep learning*.

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

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

O4. Desenvolver e testar a geração de código do Penpot 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 geração de *wireframes* de aplicativos App Inventor limitando os elementos de interface de UI (*user interface*) a estes que podem ser utilizados nos apps App Inventor. A automatização enfocará somente no desenvolvimento de projetos com a ferramenta Penpot.

1.3. Metodologia de pesquisa

A metodologia de pesquisa aplicada utilizada neste trabalho é o multi-método incluindo as 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 de apps App Inventor;

A1.2 – Síntese de conceitos de representação de projetos no Penpot;

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

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. Este mapeamento é realizado como continuação do mapeamento realizado por Baulé (2020). Esta etapa é dividida nas seguintes atividades:

A2.1 – Revisão da 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 por detecção de objetos

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 – Preparação de conjunto de imagens;

A3.3 – Seleção, treinamento e avaliação da rede neural

A3.4 – Predição/Inferência.

Etapa 4 – Desenvolvimento do código de Penpot

Nesta etapa é desenvolvido o módulo para transformar o modelo semântico em código do Penpot, 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 Penpot a partir de *sketches* usando *deep learning*. O capítulo 4 apresenta o modelo de rede neural utilizado, como foi treinada como também como foi implementado o programa que a partir do resultado da detecção cria um projeto Penpot. O Capítulo 5 apresenta conclusões como também propostas para trabalhos futuros.

2. Fundamentação teórica

2.1 *Design* de interface com App Inventor

O App Inventor (MIT, 2023) é uma ferramenta de programação visual de código-aberto, que permite a criação de aplicativos para dispositivos móveis. Desenvolvida pelo Google, com a intenção de permitir que qualquer pessoa possa criar seus próprios aplicativos, e hoje é mantido pelo Instituto de Tecnologia de Massachusetts (MIT). Com esta proposta foi criado um ambiente de desenvolvimento utilizando uma linguagem de programação visual em blocos simples que podem ser combinados para implementar comportamentos complexos. Conta com mais de um milhão de usuários mensais, em 195 países que produziram mais de 85 milhões de aplicações (MIT, 2023).

No App Inventor o desenvolvimento de aplicações é dividido em duas áreas, *Designer* e Blocos.

A área de Blocos é utilizada para fazer a programação lógica da aplicação, como funções, laços de repetição, condições, etc.

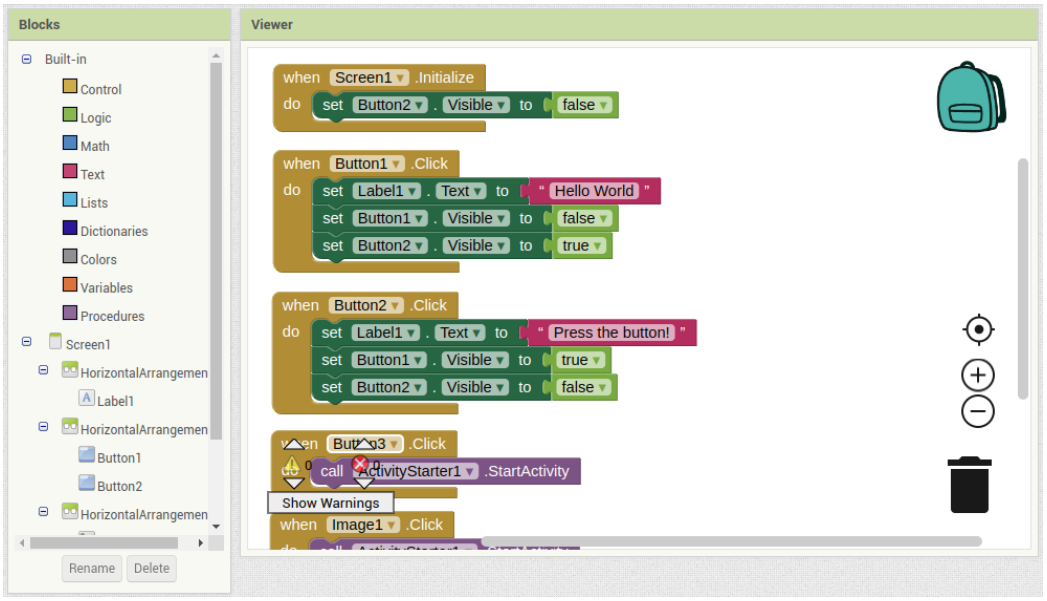


Figura 1. Exemplo área de blocos do App Inventor

Na área de *designer* são configurados os componentes da interface visual como botões, áreas de texto, entre outros.

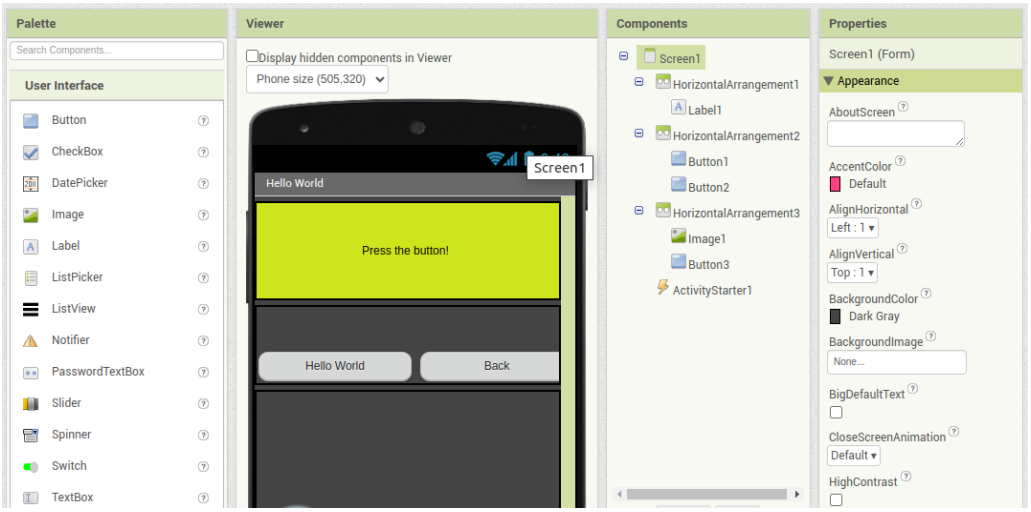


Figura 2. Exemplo área de *designer* do App Inventor

Componente	**Descrição**	**Exemplo**
Botão	Botão com a habilidade de detectar cliques.	Text for Button1
Caixa de Seleção	Caixa de seleção que pode disparar um evento ou mudar seu estado booleano quando o usuário clica nela.	Text for CheckBox1 Text for CheckBox1
Imagem	Componente para apresentação de imagens.	MIT APP INVENTOR
Legenda	Uma Legenda mostra um trecho de texto.	Text for Label1
Escolhe Lista	Um botão que, quando clicado, mostra uma lista de textos para o usuário escolher.	Element1 Element2
Visualizador De Listas	Componente para a apresentação de lista de elementos textuais.	Elemento 1 Detalhes Elemento 1 Elemento 2 Detalhes Elemento 2
Caixa De Senha	Uma caixa para digitar senhas. Mesma funcionalidade que Caixa De Texto, exceto por não exibir os caracteres digitados.	Password
Deslizador	Uma barra de processo com indicador arrastável, que pode ser ajustado pelo usuário.	

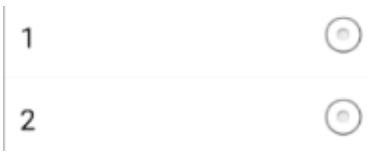
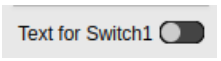
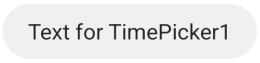
Lista Suspensa(spinner?)	Um componente para seleção que exibe um pop-up com uma lista de elementos.	
Switch	Interruptor que gera um evento ao ser clicado pelo usuário.	
Caixa De Texto	Uma caixa para o usuário inserir texto.	
Escolhe Hora	Um botão que, quando clicado, inicia um diálogo que permite ao usuário selecionar um horário.	

Tabela 1. Exemplos de elementos visíveis de *design* de interfaces de apps com App Inventor *core (Theme Default Device)*

Cada componente pode ter métodos, eventos e propriedades dependendo de seu tipo, por exemplo a caixa de texto pode-ser alterado:

- Cor de fundo
- Como o texto apresentado é formatado (itálico, negrito, tamanho, tipo e cor da fonte)
- O tamanha do componente (altura e largura)
- A visibilidade inicial

Dentre outras propriedades, várias propriedades estão presentes em diversos elementos, mas alguns elementos têm propriedades específicas, como por exemplo a Caixa de Seleção tem a propriedade de item selecionado.

2.2 Representação de projetos no Penpot

O Penpot (Kaleidos, 2023) é uma ferramenta de *design* gráfico e prototipação de código-aberto, que permite a criação de artefatos gráficos incluindo o desenvolvimento de *design* de interfaces de apps. Com o intuito de facilitar a colaboração entre *designers* e desenvolvedores, é desenvolvido e mantido por Kaleidos.

O Penpot utiliza uma estrutura de camadas que podem ser populadas por elementos simples como linhas, quadrados e círculos, que combinados formam elementos mais complexos que podem ser agrupados. Estes elementos (grupos de elementos) também podem ser guardados como um novo componente (*assets*). Nestes elementos pode ser

configurado tamanho, cor, posição, bordas e até efeitos como sombra e desfoque entre outras.

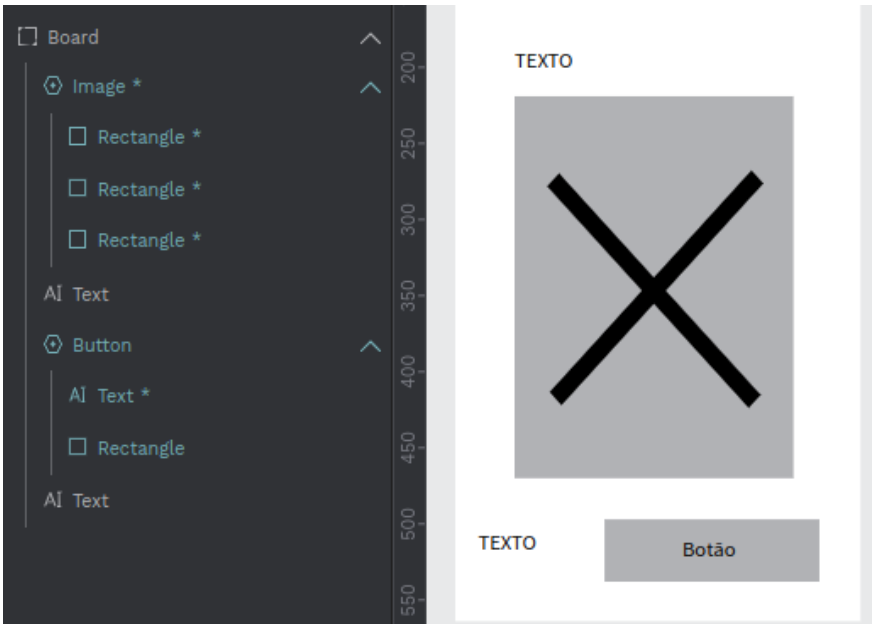


Figura 3. Exemplo de wireframe de uma interface básica

A estrutura de dados que guarda a representação dos elementos é uma árvore, que começa num nodo de página, no qual são adicionados nodos de formas, ou componentes (coleção de nodo de formas) (Figura 3).

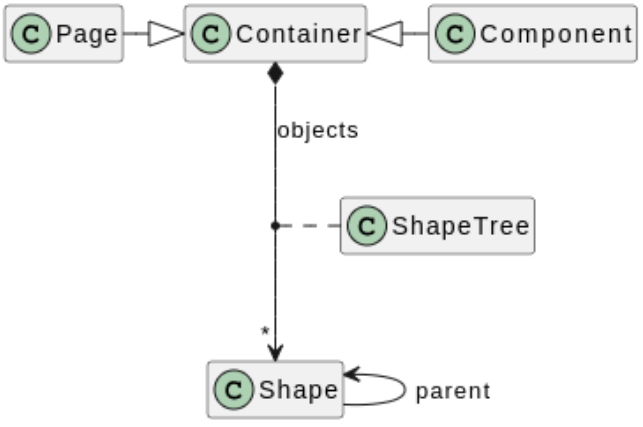


Figura 4. Estrutura de árvore dos componentes no Penpot (Kaleidos, 2023).

Formas são os elementos mais importantes da representação. São objetos abstratos que representam desde retângulos, textos e sombra (Figura 4.). Eles e suas propriedades formam grande parte dos arquivos de projeto.

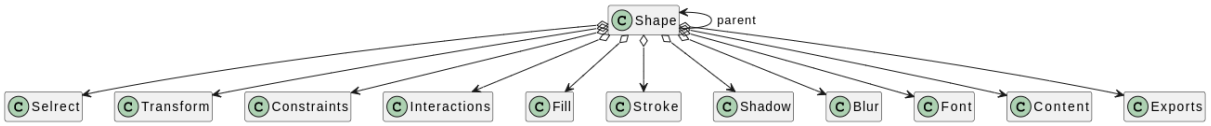


Figura 5. Formas e suas derivações no Penpot (Kaleidos, 2023).

Projetos podem ser exportados e importados como uma combinação de arquivos .svg e .json ou no formato proprietário .penpot. Nestes arquivos são descritas as características de cada elemento além dos recursos que os formam.

O .json, nomeado "*manifest.json*" e presente no diretório raiz, é utilizado para guardar informações sobre o projeto, por exemplo o nome, arquivos e suas páginas (Figura 6.). Elementos são identificados por um id único, e páginas são descritas no .svg de mesmo nome.

```
{
    "teamId": "", // id da equipe criadora
   "fileId": "", // id do projeto
   "files": {
      "$fileId": { // propriedades do projeto, identificado pelo id
         "features": [], // lista de
         "libraries": [], // lista de bibliotecas adicionais
         "hasDeletedComponents": false,
         "hasComponents": true,
         "name": "Sketch2Penpot", // nome do projeto
         "pagesIndex": { // lista índices das páginas
            "$PageId": { // índice único da página
               "name": "Page 1" // nome da página
            }
         },
         "exportType": "all",
         "pages": [ // lista dos índices de página em formato de string
            "$PageId"
         ],
         "hasMedia": false,
         "shared": false,
         "version": 2,
         "hasTypographies": false,
         "hasColors": false
      }
   }
}
```

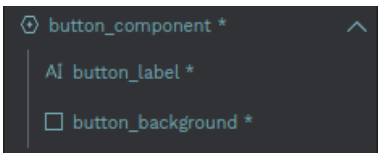
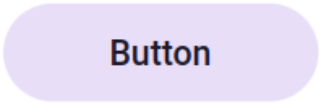
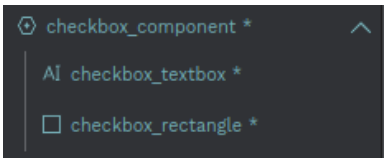
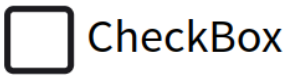
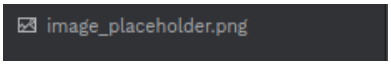
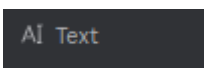
Figura 6. Estrutura do *manifest.json*

No .svg cada forma é descrita em uma tag, podendo ou não ter propriedades adicionais identificadas com Penpot (Figura 7). Estas propriedades são utilizadas pelo importador para reconstruir o projeto, e caso não estejam presentes serão inferidos (Kaleidos, 2023).

```
<g id="shape-35731e47-4cd9-8096-8002-56d99e5197bf">
<penpot:shape penpot:name="Rectangle" penpot:type="rect"
    penpot:transform="matrix(1.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000)"
    penpot:transform-inverse="matrix(1.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000)"
    penpot:proportion="1" penpot:proportion-lock="false" penpot:rotation="0" penpot:center-x="400"
    penpot:center-y="352.50000000000017">
    <penpot:fills>
        <penpot:fill penpot:fill-color="#a9afec" penpot:fill-opacity="1">
        </penpot:fill>
    </penpot:fills>
    <penpot:layout-item penpot:layout-item-h-sizing="fix" penpot:layout-item-v-sizing="fix">
    </penpot:layout-item>
</penpot:shape>
<defs>
</defs>
<g class="fills" id="fills-35731e47-4cd9-8096-8002-56d99e5197bf">
    <rect rx="0" ry="0" style="fill:#a9afec;fill-opacity:1" x="200" y="83.00000000000011"
        transform="matrix(1.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000)" width="400"
        height="539.0000000000001">
    </rect>
</g>
</g>
```

Figura 7. Exemplo da descrição de um retângulo.

A Tabela 2 descreve os componentes que são considerados no escopo do presente trabalho e sua composição e aparência.

Componente	**Estrutura no penpot**	**Exemplo**
Botão		
Caixa de Seleção		
Imagem		
Legenda		

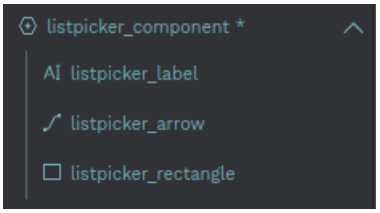
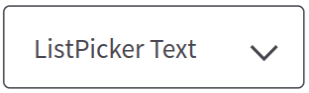
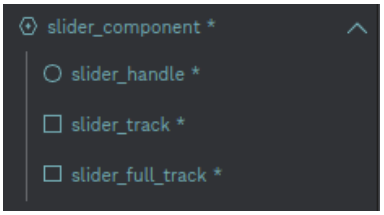
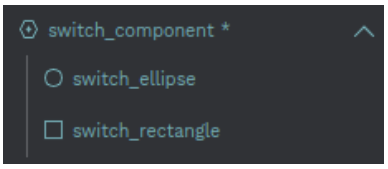
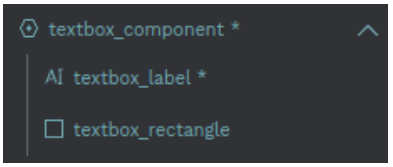
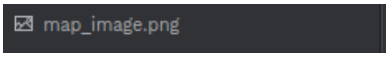
Escolhe Lista		
Deslizador		
Switch		
Caixa De Texto		
Mapa		

Tabela 2. Exemplos de elementos de interface básicos criados no Penpot.

2.3 Detecção de objetos com *Deep Learning*

Inteligência artificial é um campo da ciência de computação que tem como foco o estudo e desenvolvimento de técnicas e sistemas que consigam executar tarefas que alcancem objetivos de maneira análoga a humanos, como reconhecimento de fala, tomada de decisão, visão computacional, entre outras (Russel; Norvig, 2009).

Redes neurais são uma dessas técnicas, inspirada pelo funcionamento do cérebro humano. Elas são compostas de várias camadas de neurônios artificiais interligadas entre si, podendo ser treinada ajustando os pesos de cada ligação, de maneira a fazer o sistema

reconhecer padrões. Elas tipicamente são adotados para tarefas na área de visão computacional incluindo o reconhecimento e classificação de imagens e detecção de objetos (Russel; Norvig, 2009).

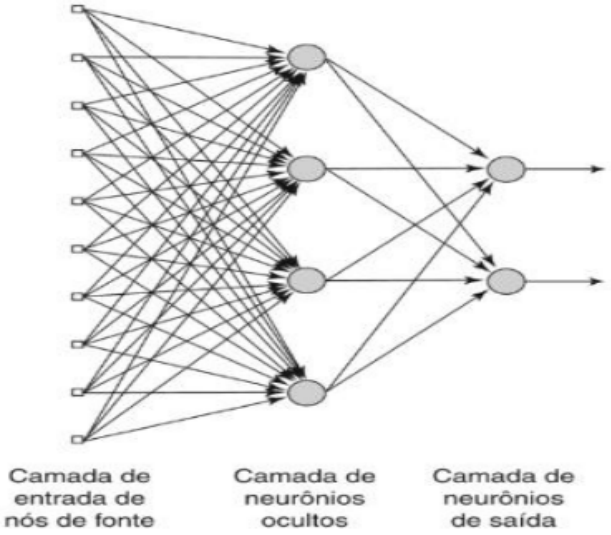


Figura 8. Estrutura básica de uma rede neural (REF)

Deep learning se diferencia de outras redes neurais por sua profundidade, ou seja a quantidade de camadas intermediárias. Cada camada é treinada a partir de um conjunto de características específicas baseadas na saída da camada anterior. As camadas intermediárias da rede neural aprendem a representação dos dados em níveis de abstração cada vez mais altos, permitindo que o sistema possa extrair características complexas dos dados (LeCun, 2023).

Para a detecção de objetos, uma das técnicas utilizadas são Redes Neurais Convolucionais (CNN) (Goodfellow, 2016). Estas são compostas por mais de uma camada convolucional, que vão detectar características da imagem, como por exemplo bordas, cores, texturas, por camadas de *pooling*, que são responsáveis por reduzir a dimensionalidade dos dados de entrada e entre as camadas de convolução, e também por camadas totalmente conectadas que fazem a detecção dos objetos na imagem (Hui, 2018).

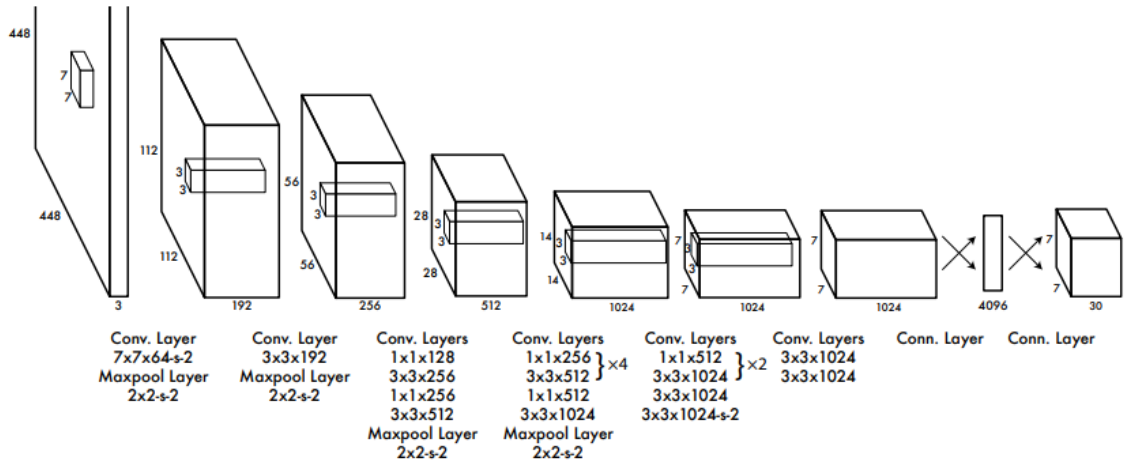


Figura 9. Exemplo da estrutura de uma Rede Neural Convolucional YOLO (*You Only Look Once*) (Hui, 2018)

Em detecção de objetos normalmente a imagem é dividida em regiões para otimizar o processo de detecção. Destas regiões são identificadas as *bounding boxes*, que são retângulos que delimitam possíveis objetos na imagem, que depois com as probabilidades de cada classe, se tornaram delimitadoras de uma determinada classe do modelo (Figura 10).

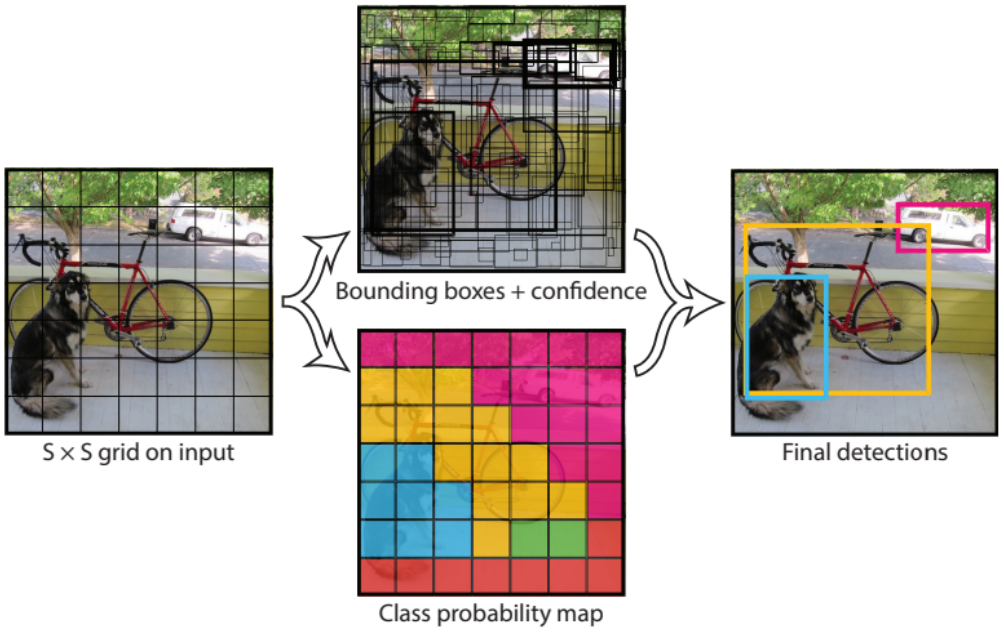


Figura 10. Modelo de visualização do processo de detecção de objetos (Brahmbhatt, 2019)

Tipicamente técnicas de detecção de objetos usam um de dois modelos: detecção de objetos em dupla etapa, ou detecção de objetos em etapa única. Modelos de detecção de etapa única, como por exemplo YOLO, são divididos em três componentes:

- *Backbone* - é uma rede pré-treinada usada para extrair as características da imagem, isto ajuda a reduzir a representação espacial da imagem como também aumenta a resolução das características extraídas.
- Pescoço (*Neck*) - é a estrutura usada para criar um pirâmide com as características, de maneira a ajudar o modelo a reconhecer objetos de diferentes escalas e tamanhos.
- Cabeça - é a parte final do modelo onde são aplicadas as âncoras no mapa de características, criando assim as classes, *bounding boxes* e as pontuações de pertencimento.

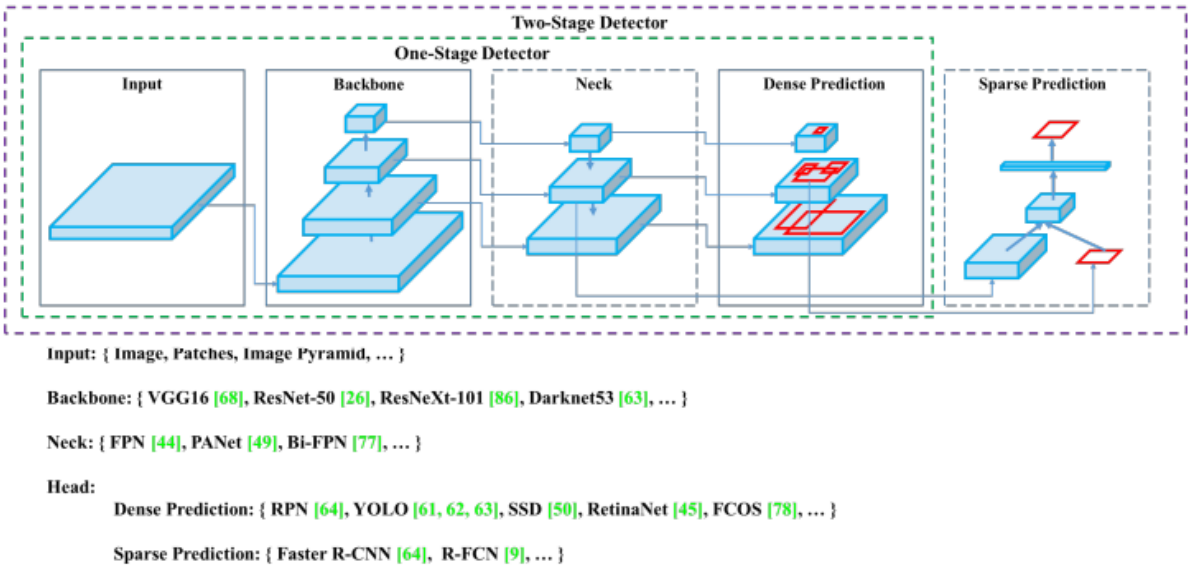


Figura 11.Detecção de objetos (Bochkovskiy, 2020)

Dentre as diversas CNNs para detecção de objetos, uma das principais é a família YOLO (*You Only Look Once*)(Córdova-Esparza, 2023). Estes modelos foram originalmente desenvolvidos por Joseph Redmon em 2015, e a versão mais atual é YOLOv8 desenvolvida pela Ultralytics.

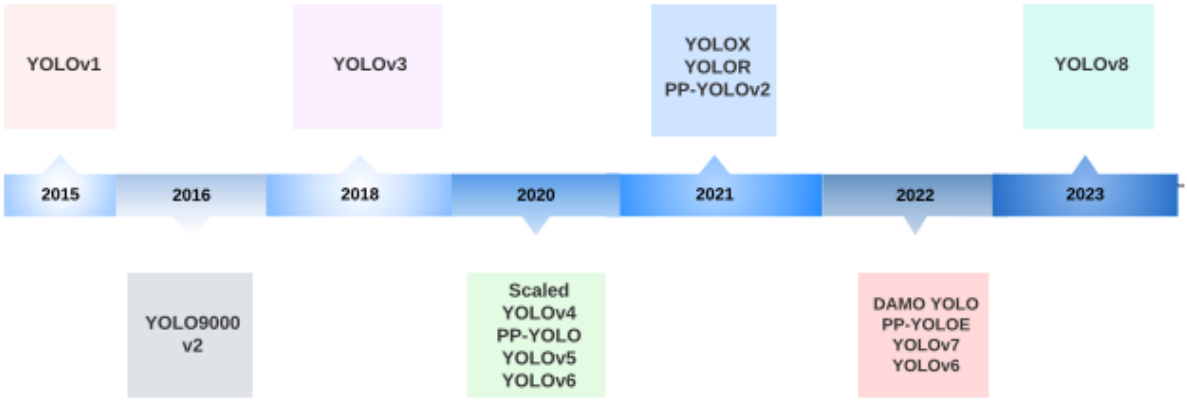


Figura 12. Linha do tempo das versões do YOLO (Córdova-Esparza, 2023)

A família YOLO é conhecida pelo balanceamento entre velocidade e precisão, é muito usada em detecção de objetos em tempo real. Dentre todas as versões suas principais características são:

- Âncoras: O modelo original YOLO não utilizava âncoras, mas YOLOv2 implementou o que aumentou a precisão das previsões das *bounding boxes.* Depois de 5 anos YOLOX introduziu uma abordagem que sem âncoras que se tornou estado da arte, e foi adotada por versões subsequentes.
- *Framework*: Darknet foi o *framework* usado nas primeiras versões, até que a Ultralytics portou YOLOv3 para Pytorch o que se tornou o padrão para as próximas versões.
- *Backbone*: A arquitetura da *backbone* da YOLO mudou significativamente durante o tempo, como utilizava o *framework* Darknet, era composto de convoluções simples e camadas de *pooling* totalmente conectadas. Com YOLOv4 foi implementado um modelo de *cross-stage* parcialmente conectado (CSP).

YOLOv5 foi lançado em 2020, e foi desenvolvido pela Ultralytics (Ultralytics, 2023). Sua estrutura é baseada em YOLOv4, sendo as principais diferenças, a mudança de *framework* para Pytorch e o uso de .yaml para configurações (Gutta, 2021). Também utiliza as técnicas de aumento de dados como mosaico e de treinamento auto adversário (SAT) introduzidas em YOLOv4 (Bochkovskiy, 2020)(Figura 13). Mosaico combina quatro imagens em grade 2 por 2, de maneira a formar uma nova imagem que será utilizada na fase de treinamento. Treinamento auto adversário (SAT) tem duas fases, na primeira a imagem é

alterada, utilizando como por exemplo *blur*, mudança de saturação, *cutmix* dentre outras, e na segunda fase a rede é treinada com a imagem alterada (Bochkovskiy, 2020).

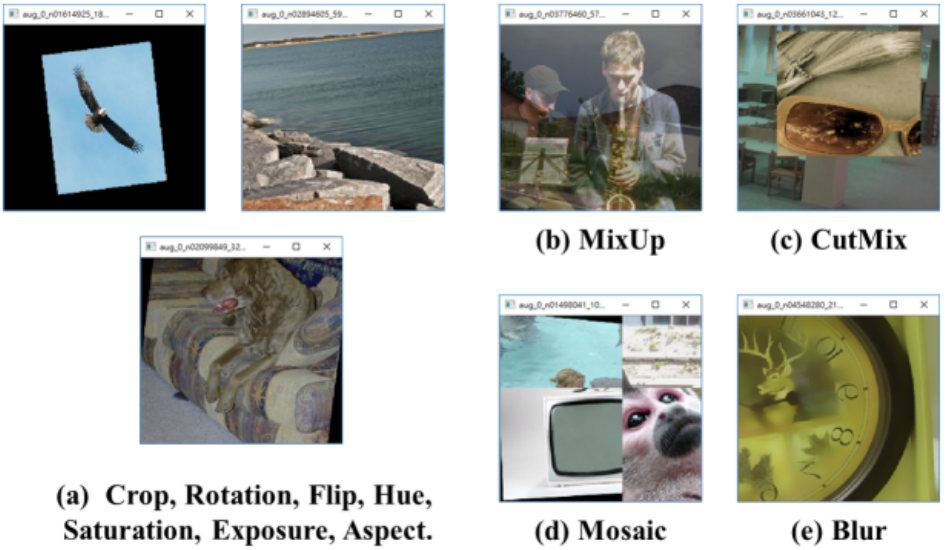


Figura 13. Técnicas de aumento de dados utilizada pela YOLO (Bochkovskiy, 2020)

YOLOv5 é disponibilizado em 5 tamanhos. Modelos maiores produzem resultados melhores, mas necessitam de mais memória CUDA e são mais lentos no treinamento. Modelos como YOLOv5 n/s são recomendados para aplicações móveis, enquanto para aplicações *cloud* são recomendadas as versões maiores como YOLOv5 l/x (Ultralytics, 2023).

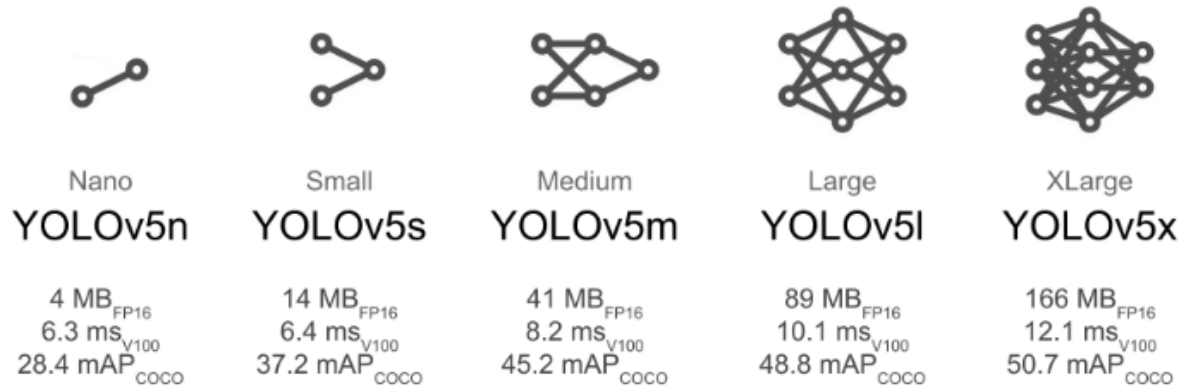


Figura 14. Diferença entre os tamanhos de versões YOLOv5 (Ultralytics, 2023)

YOLOv7 foi publicado em 2022, e se assemelha a YOLOv4-scaled (Wang, 2022). As principais diferenças na arquitetura são o uso de uma rede de agregação de camada eficiente estendida (E-ELAN) e o dimensionamento para modelos baseados em concatenação. Também foram adicionadas algumas *bag of freebies* treináveis (Kukil, 2022,

Boesch, 2023). *Bag of freebies* é o termo utilizado para definir métodos de treinamento que aumentam a precisão da rede, sem aumentar o tempo de resposta da rede pós treinada. Em detecção de objetos, essas técnicas normalmente visam o aumento do espaço de treinamento, sem aumentar a base de imagens de treinamento (Bochkovskiy, 2020).

Uma das técnicas adotadas em YOLOv7, foi o uso de múltiplas cabeças, sendo que uma delas é a cabeça principal que faz a previsão baseada nas *labels* da imagem de entrada, enquanto as cabeças auxiliares usam a previsões da cabeça principal para gerar as suas, estas cabeças utilizam pedaços maiores da imagem aumentando assim a geração de reconhecimentos positivos, o esperado deste comportamento é que a cabeça principal não perca informações importantes a serem aprendidas (Wang, 2022) .

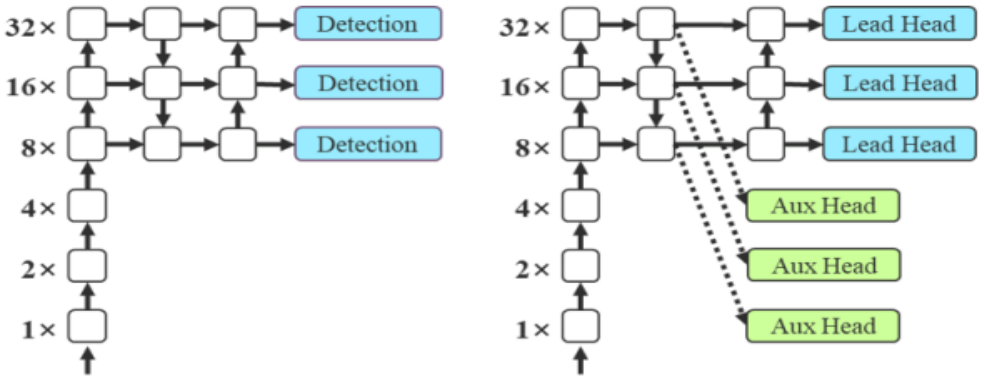


Figura 15. Diferença entre modelo de cabeça única, e modelo com cabeças auxiliares (Wang, 2022)

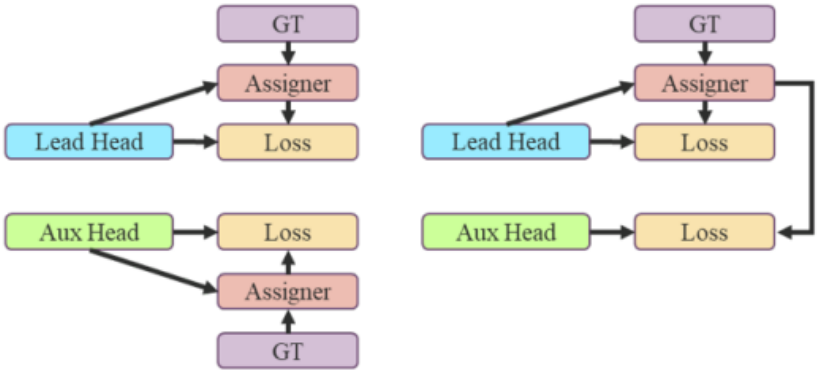


Figura 16. Diferença entre modelo em que cabeças auxiliares são treinadas como a cabeça principal e modelo em que é usado a previsão da cabeça principal para treinar as auxiliares (Wang, 2022)

YOLOv8 foi lançado em 2023, desenvolvido e mantido pela Ultralyrics. É disponibilizada em 5 tamanhos (Tabela 3) e aumentou significativamente a velocidade e precisão das soluções (Figura 17). Esta nova versão também aumentou a facilidade de uso, diminuindo o número de parâmetros necessários para a configuração. Também foi adicionada uma API em Python como também uma linha de comando.

Modelo	tamanho (pixels)	mAP^{val} 50-95	Velocidade CPU ONNX (ms)	Velocidade A100 TensorRT (ms)	parâmetros (M)	FLOPs (B)
YOLOv8n	640	37,3	80,4	0,99	3,2	8,7
YOLOv8s	640	44,9	128,4	1,20	11,2	28,6
YOLOv8m	640	50,2	234,7	1,83	25,9	78,9
YOLOv8l	640	52,9	375,2	2,39	43,7	165,2
YOLOv8x	640	53,9	479,1	3,53	68,2	257,8

Tabela 3. Modelos YOLOv8 treinadas e validadas com conjunto COCO (Ultralytics, 2023).

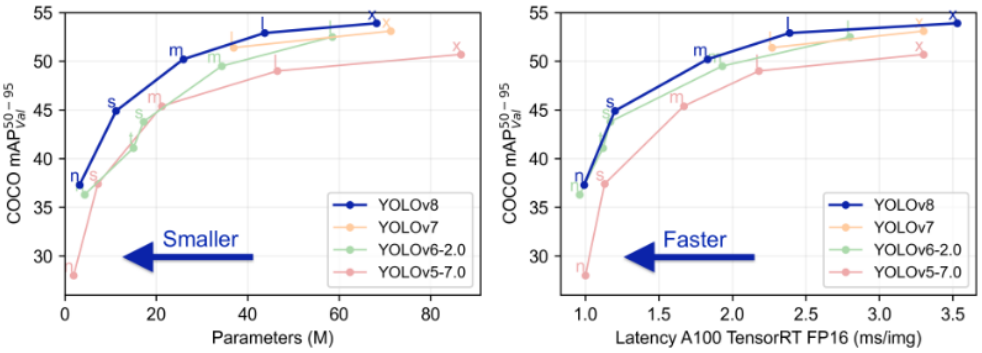


Figura 17. Comparação entre redes da família YOLO, usando conjunto COCO (Ultralytics, 2023).

2.2.5 Avaliação de desempenho de modelos de detecção de objetos

Na avaliação de desempenho de modelos de detecção de objetos tipicamente é utilizado IOU (*intersection over union*), uma medida baseada em índice de Jaccard. Esta medida é obtida a partir da divisão da intersecção e união entre a *bounding box* predita e a *bounding box* verdadeira (Goodfellow, 2016)(Figura 18). Este valor varia entre 0 e 1, e é definido um valor de corte que será utilizado para considerar se uma detecção correta ou não, normalmente é utilizado um valor de 0,5 nas fases de treino.

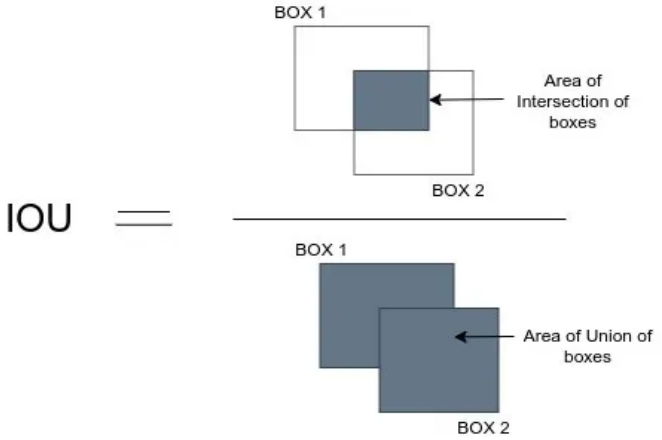


Figura 18. Representação visual do IOU (*intersection over union*)(Subramanyam, 2021).

Ao utilizar o modelo IOU, os resultado podem ser classificados da seguinte maneira:

- Positivo Verdadeiro (PV): detecção correta com IOU > que valor de corte
- Falso Positivo (FP): detecção errado com IOU < que valor de corte
- Falso Negativo (FN): a *bounding box* verdadeira não foi detectada

Dependendo do contexto e modelo a ser avaliado outras métricas podem ser utilizadas, a Tabela 3 resume algumas das métricas mais utilizadas.

Categoria	**Métrica**	**Definição**
Métricas de Classificação	Acurácia	Número de predições corretas dividido pelo número total de predições, multiplicado por 100
	Precisão	Precisão = PV / (PV + FP)
	Recall	*Recall* = PV / (PV + FN)
	Pontuação F1	F1 = 2 * Precisão * *Recall* / (Precisão + *Recall*)
Métricas de Regressão	Erro quadrático médio	A média do quadrado da diferença entre o valor predito e o valor verdadeiro
	Erro quadrático absoluto	A média absoluta da diferença entre o valor predito e do valor verdadeiro

Tabela 4. Exemplo de métricas utilizadas na avaliação de modelos de detecção de objetos (Baulé, 2020).

Nos últimos anos, a precisão média (*average precision* (AP)) é uma métrica que tem sido frequentemente usada (Goodfellow, 2016). O AP para cada classe é a área sob a curva de acurácia-*recall*, no qual o *recall* no eixo x e a acurácia é plotada no eixo y, e seu valor varia entre 0-1, sendo que valores maiores indicam melhor desempenho (Goodfellow,

2016). É usada a média entre as classes (mAP) para comparar diferentes modelos de visão computacional, valores acima de 0.5 são considerados bons para a maior parte das tarefas. Tendo valores de mAP altos como objetivo, por exemplo o mAP 90, torna o sistema mais preciso mas poderá refletir métricas menores, pois detecções terão mais dificuldade a atingir o valor de corte, enquanto que utilizar valores de mAP mais próximos de 50 (normalmente valor mínimo aceito como razoável) possivelmente aumentará as métricas de identificação, mas gerará um maior número de erros, como falsos positivos. É importante escolher o valor de mAP apropriado com base nos requisitos da aplicação e na importância da precisão das detecções em relação à confiança das mesmas. Figura 18 utiliza esta métrica para comparar diferentes versões de YOLO.

Version	Date	Anchor	Framework	Backbone	AP (%)
YOLO	2015	No	Darknet	Darknet24	*63.4*
YOLOv2	2016	Yes	Darknet	Darknet24	*63.4*
YOLOv3	2018	Yes	Darknet	Darknet53	36.2
YOLOv4	2020	Yes	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Yes	Pytorch	Modified CSP v7	55.8
PP-YOLO	2020	Yes	PaddlePaddle	ResNet50-vd	45.9
Scaled-YOLOv4	2021	Yes	Pytorch	CSPDarknet	56.0
PP-YOLOv2	2021	Yes	PaddlePaddle	ResNet101-vd	50.3
YOLOR	2021	Yes	Pytorch	CSPDarknet	55.4
YOLOX	2021	No	Pytorch	Modified CSP v5	51.2
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet	54.7
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	RepConvN	56.8
DAMO-YOLO	2022	No	Pytorch	MAE-NAS	50.0
YOLOv8	2023	No	Pytorch	YOLO v8	53.9

Figura 19. Comparação das diferentes versões de YOLO (Córdova-Esparza, 2023)

3. Estado da arte

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*?

Observando que já foi realizada uma revisão da literatura com este enfoque em outubro de 2019 (Baulé, 2020) o presente trabalho realiza uma atualização deste mapeamento no período de 2019-2023. Assim, para esse mapeamento, são mantidas as mesmas 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.

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* feitos por uma ferramenta de *software* no lugar de *sketches* manuais.
- Para tornar a pesquisa mais ampla, foram incluídos também artefatos que geram uma saída diferente de *wireframes* em Penpot, incluindo representações intermediárias da estrutura da tela, códigos etc.

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

***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 5). Usou se também sinônimos para minimizar o risco de omissão de trabalhos relevantes. Neste mapeamento foram mantidos os mesmos termos de busca e *string* de busca em conformidade com Baulé (2020).

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

Tabela 5 - Termos de busca e respectivos sinônimos

(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”)

Foi realizada sua adaptação para as diferentes bases de dados consideradas e para o período a ser considerado (Tabela 6).

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 > 2019
arXiv.org e-print archive	order: -announced_date_first; size: 50; date_range: from 2019-10-01 to 2023-12-31; classification: Computer Science (cs); 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"

Tabela 6 - Strings de buscas utilizadas nas diferentes bases de dados

3.2 Execução da busca

A busca dos artigos foi realizada em maio de 2023 pelo autor do presente trabalho e revisada pela orientadora. A busca inicial resultou em 117 artigos, dos quais foram selecionados artigos relevantes de acordo com os critérios de inclusão, exclusão e qualidade (Tabela 7).

Base de Dados	**Quantidade de artigos resultantes da busca**	**Quantidade de artigos relevantes**
ACM Digital Library	20	2
IEEE Xplore Digital Library	36	2
Scopus	82	5
arXiv.org e-print archive	19	3
Total (sem duplicados)	**117**	**7**

Tabela 7 - Resultados da Busca

Foram lidos os títulos e resumos dos artigos encontrados na busca para fazer uma primeira seleção de potenciais artigos relevantes. Estes foram lidos por completo para determinar sua relevância utilizando os critérios de qualidade. Alguns dos artigos foram encontrados em mais de uma base de dados. Depois da aplicação dos critérios de seleção e remoção de repetidos foram identificados 7 artigos relevantes (Tabela 7).

3.3. Análise dos Resultados

Visando complementar a revisão da literatura realizada em 2019 (Baule, 2020), foram utilizadas as mesmas categorias para extração dos dados (Tabela 8).

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.
PA4. Quais conjuntos de dados usaram e quais as características desses conjuntos?	Descrição	Descrição do *Dataset*
	Quantidade de instâncias	O número de instâncias de dados presente no dataset.
PA5. Qual tipo de rede usaram e quais as características dessa	Qual o modelo de rede utilizado?	Detalhes sobre a Rede utilizada
	Tipo de aprendizagem	Tipo de aprendizagem (supervisionada, não-supervisionada etc.)
PA6. 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

Tabela 8 - Informações extraídas.

Os artigos selecionados foram lidos por completo e extraídas as informações relevantes, caso não esteja presente no artigo será identificada como não informada (NI).

PA1. Quais modelos/ferramentas existem?

Foram identificados 7 trabalhos que apresentam a geração de *wireframes* de interface, a partir de *sketches* utilizando alguma técnica de Machine Learning (Tabela 9).

Citação	Referência Bibliográfica
(Abdelhamid, 2020)	A. A. Abdelhamid; S. R. Alotaibi; A. Mousa; **Deep learning-based prototyping of android GUI from hand-drawn mockups**. The Institution of Engineering and Technology, 2020
(Barua, 2022)	S.S.Barua; I. M. Zulkarnain; A. R.; G.R. Alam; Z. Uddin; **Sketch2FullStack: Generating Skeleton Code of Full Stack Website and Application from Sketch using Deep Learning and Computer Vision**. arXiv, 2022
(Baulé et al., 2021a) (Baulé et al., 2021b)	D. Baulé, C.Gresse von Wangenheim, A. von Wangenheim, J. C. R. Hauck, E. C. Vargas Júnior; **Automatic code generation from sketches of mobile applications in end-user development using Deep Learning.** arXiv, 2021 D. Baulé, C.Gresse von Wangenheim, A. von Wangenheim, J. C. R. Hauck, E. C. Vargas Júnior; **Using Deep Learning to Support the User Interface Design of Mobile Applications with App Inventor.** IHC '21: Proceedings of the XX Brazilian Symposium on Human Factors in Computing Systems, Brasil, 2021
(Jain, 2019)	V. Jain; P. Agrawal; S. Banga; R. Kapoor; Shashwat Gulyani; **Sketch2Code: Transformation of Sketches to UI in Real-time Using Deep Neural Network.** arXiv, 2019
(Moran, 2020)	K. Moran; C. Bernal-Cardenas; M. Curcio; R. Bonett; D. Poshyvanyk; **Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps.** IEEE Transactions on Software Engineering, 2020
(Sun, 2020)	X.Sun, T. Li, J.Xu; **UI Components Recognition System Based On Image Understanding.** IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Macau, China, 2020
(Wimmer, 2021)	C. Wimmer, A. Untertrifaller, T.Grechenig; **SketchingInterfaces: A Tool for Automatically Generating High-Fidelity User Interface Mockups from Hand-Drawn.** OzCHI '20: Proceedings of the 32nd Australian Conference on Human-Computer Interaction, Sydney, Australia, 2021

Tabela 9 - Artigos relevantes publicados no período de 2019-2023.

Podemos ver que a geração automática de *wireframes* continua uma preocupação entre os pesquisadores.

PA2. Quais os dados de entrada/saída?

Observando os resultados extraídos, foi possível verificar uma grande variação nos formatos de saída (Tabela 10).

Citação	Dados			
	Dados de Entrada	**Formato dos dados de Entrada**	**Dados de Saída**	**Formato dos dados de Saída**
(Abdelhamid, 2020)	*Sketch*	Imagens	Código	XML
(Barua, 2022)	*Sketch*	Imagens	Código	Json, Python, SQL
(Baulé, 2021)	*Sketch*	Imagens	Projeto para App Inventor	.aia

(Jain, 2019)	*Sketch*	Imagens	Código da UI	HTML
(Moran, 2020)	*Screenshots*	Imagens	Código	Java, XML
(Wimmer, 2021)	*Sketch*	Imagens	Código	HTML
(Sun, 2020)	*Screenshots*	Imagens	Imagem com elementos de interface rotulados	Imagem

Tabela 10 - Informações sobre dados de entrada e saída extraídos.

Dentre os artigos analisados, houve uma clara divisão entre os formatos de entrada, sendo que apenas dois utilizando *screenshots* de aplicações existentes, Moran et al. (2020) e Sun et al. (2020). Dentre os oito que utilizaram imagens de *sketches* como entrada, Wimmer et al. (2021) se destaca por ter utilizado imagens capturadas por uma câmera em tempo real.

Em relação aos dados de saída, todos os artigos analisados fornecem uma representação da interface em código, exceto Sun et al. (2020) que retorna apenas a imagens com os componentes identificados.

Jain et al. (2019) e Wimmer et al. utilizam HTML e Abdelhamid et al. XML, linguagens que são multiplataforma. Baulé et al. (2021) focou na geração de código para App Inventor. Moran (2020) e Baura (2022) foram além de criar a representação visual da interface, e criaram também estrutura de classes para o *back-end* da aplicação na linguagem designada. Baura (2022) por se propor a criar uma aplicação *full-stack* também cria a estrutura básica das classes em SQL, porém os componentes visuais são apenas descritos em um Json.

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

Os artigos analisados apresentam um foco em aplicativos mobile e em seus elementos mais encontrados (Tabela 10).

Citação	**Dados**		
	Elementos detectados	**Tipo de software**	**Plataforma**
(Abdelhamid, 2020)	*ImageView, ListView, Button, EditText, ProgressBar, ComboBox, Switch, SliderBar, CheckBoxOn, CheckBoxOff, RadioOn, RadioOff, TextView*	Aplicativo	Android
(Barua, 2022)	*Checkbox, Textbox, Button, Label, Radio Buttons, Heading, Combobox, Link, Image,*	Aplicativo	NI

	Paragraph.		
(Baulé, 2021)	*Label, Button, Image, TextBox, CheckBox, ListPicker, Slider, Switch, Map.*	Aplicativo	App Inventor
(Jain, 2019)	*Checkbox, Textbox, Button, Label, Radio Buttons, Heading, Combobox, Link, Image e Paragraph.*	Diversos	*Website, Mobile*
(Moran, 2020)	*ImageView, ImageButton, Button, Spinner, RatingBar, NumberPinker, ToggleButton, SeekBar, Switch, ProgressBar, RadioButton, CheckBox, CheckedTextView, EditText*	Aplicativo	*Mobile*
(Wimmer, 2021)	*button, input field, image, text, info card, carousel, navigation bar and table.*	Aplicativo	*Website/ Mobile*
(Sun, 2020)	*Button, CheckBox, CHeckTextView, EditText, ImageButton, ImageView, ProgressBarHorizontal, ProgressBarVertical, RadioButton, RatingBar, Switch, SeekBar, Spinner, TextView.*	Aplicativo	Android

Tabela 11 - Elementos de interface considerados.

Podemos constatar por meio da extração dos dados que existe um foco nos componentes mais comuns, identificando no máximo 14 elementos diferentes. Baulé et al. (2021) é o único que cita um estudo de terceiros, com enfoque nesta análise, como sua fonte e Moran et al. (2020) o único que executou pesquisa própria para determinar seus componentes.

Dentre os artigos analisados, 6 geram código para aplicações *mobile,* sendo que Sun et al. (2020) e Abdelhamid et al. (2020) geram especificamente para plataforma Android, e Baulé et al. (2021) para App Inventor.

Jain et al. (2019) e Wimmer et al. (2021) utilizam uma codificação intermediária em formato Json, permitindo que sejam implementados tradutores para diferentes plataformas. Ambos utilizam um tradutor para HTML e visualizador web para observar os resultados.

Barua et al. (2022) além dos componentes também faz a extração e identificação do texto, que é usado no processo de criação de classes para os componentes.

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

Todos os artigos que foram analisados utilizam conjuntos proprietários criados pelos autores (Tabela 12).

	Nome do conjunto de dados	Quantidade de imagens
(Abdelhamid, 2020)	Conjunto proprietário	490 *sketches*
(Barua, 2022)	Pré-treinada com ImageNet, Conjunto proprietário	500 *sketches* com 5899 elementos de interface
(Baulé, 2021)	Conjunto proprietário	237 *sketches*
(Jain, 2019)	Pré-treinada com COCO dataset, Conjunto proprietário	149 *sketches* com 2001 elementos de interface
(Moran, 2020)	Conjunto proprietario	19,786 *screenshots* de 8,878 apks diferentes
(Wimmer, 2021)	Conjunto proprietário	De 6 à 10 *sketches* de cada componente a ser identificado
(Sun, 2020)	Conjunto proprietário	NI

Tabela 12 - Características dos conjuntos de dados.

Moran et al. (2020) desenvolveram um software para extrair dados como posição e tamanho dos metadados de aplicativos Android utilizados para identificar elementos de interface em *screenshots* de maneira a criarem seu *dataset* de treinamento. Foram coletados e rotulados 19786 screenshots de 8878 aplicativos gratuitos diferentes de 39 categorias.

Sun et al. (2020) utiliza *screenshots* como entrada para seu modelo, e utilizou trabalhos como o de Moran et al. (2020) para montar seu conjunto de dados de treinamento, com um número não informado de elementos.

Os outros artigos analisados focam na identificação de elementos em *sketches*, e por isso criaram e rotularam seus próprios conjuntos, com isto podemos ver que seus conjuntos são menores que os de Moran et al. (2020).

Wimmer et al. (2021) utilizou DoodleClassifier para identificar os elementos de UI, e reportaram que com 6 à 10 *sketches* e cada componente foi o suficiente para um resultado aceitável para suas necessidades.

Baura et al. (2022) e Jain et al. (2019) utilizaram redes pré-treinadas para diminuir o tempo de treinamento.

Abdelhami et al. (2020) tem uma abordagem diferente dos outros, em vez de criar *sketches* que tentam representar interfaces reais, fazem *sketches* que contém todos os elementos que pretendem identificar, de maneira que o conjunto de treinamento tenha a mesma quantidade de cada elemento.

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

Pelo conjunto de dados obtidos podemos observar uma clara preferência por CNNs para a tarefa de identificação dos componentes de interface. Também podemos observar a preferência no uso de redes pré-treinadas, o que torna o tempo de treinamento menor (Tabela 13).

Citação	**Treinamento**	
	Modelo de Rede	**Tipo de aprendizagem**
(Abdelhamid, 2020)	CNN (Yolov5)	Rede pré-treinada, Supervisionada
(Barua, 2022)	Faster RCNN with Inception and Resnet V2 (Deteção de elementos) EAST (deteção do texto)	Rede pré-treinada com COCO, Supervisionada
(Baulé, 2021)	CNN (YOLO)	Rede pré-treinada com ImageNet, Supervisionada
(Jain, 2019)	RetinaNet (ResNet + *Feature Pyramid Network*)	ResNet pré-treinada em ImageNet, Supervisionada
(Moran, 2020)	CNN (Baseada em AlexNet)	Supervisionada
(Wimmer, 2021)	CNN (*doodle classifier*)	NI
(Sun, 2020)	CNN	Supervisionada

Tabela 13 - Informações sobre as redes neurais utilizadas.

Abdelhamid et al. (2020) e Baulé et al. (2021) utilizaram redes YOLO pré-treinadas em seus projetos. Ambos utilizam uma saída em Json como entrada para o processo de geração final do código. Uma diferença no trabalho de Abdelhamid et al. (2020) dos outros trabalhos apresentados, é o processo de alinhamento dos componentes antes da geração final.

Barua et al. (2022) é o único que além de identificar os elementos de interface, também utiliza EAST, um recurso da biblioteca OpenCV, para identificar os textos presentes nos *sketches*. Com os outputs destes dois processos são criadas descrições em Json das tabelas de classe para cada componente que depois são usadas na geração da interface gráfica de usuário, como também classes básicas nas linguagens Python e SQL, que seriam utilizadas no *back-end* da aplicação.

Jain et al (2019) utiliza uma estrutura própria, utilizando camadas ResNet pré-treinadas com ImageNet, seguidas por camadas FPN (*Feature Pyramid Network*) para aumentar a qualidade dos dados obtidos. Uma preocupação levada em conta por Jain et al (2019), é a possibilidade do output do processo de detecção dos elementos, resulte em componentes sobrepostos, foi criado um algoritmo que trata desses casos.

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

Podemos observar uma variação nos métodos de avaliação adotados entre os artigos analisados, sendo que a maioria adotou alguma medida matemática baseada nos resultados das predições dos modelos (Tabela 14).

Citação	**Avaliação do desempenho**		
	Medidas de desempenho	**Avaliação do modelo**	**Resultados da avaliação**
(Abdelhamid, 2020)	Utilizadas as medidas precisão, *recall* e pontuação F1. Comparada acurácia com outros modelos	Utilizados 100 dos *sketches* desenvolvidos para o teste.	Precisão, recall e pontuação F1 maior ou igual a 0.96. Acurácia média de 98,54, maior entre os modelos considerados na comparação.
(Barua, 2022)	Utilizadas as medidas mAP, média de *recall* e perda total.	Utilizados um número não informado de *sketches* desenvolvidos.	mAP 50 de 0.9944, mAP 75 de 0.8845, média de *recall* de 0.8323 e perda total de 0.2488
(Baulé, 2021)	Questionário de avaliação com participantes voluntários	Foram gerados 10 *wireframes* a partir de *sketches* randômicos do conjunto de teste para serem avaliados. Os participantes criaram seus *sketches* e testaram a aplicação.	De acordo com os participantes, a maioria dos *wireframes* gerados representavam totalmente ou quase totalmente os *sketches* associados.
(Jain, 2019)	NI	NI	NI
(Moran, 2020)	Precisão do modelo. Precisão de hierarquia de componentes. Semelhanças visual. Aplicação industrial.	Usados 10% do conjunto de dados para teste. Extraídas 83 imagens para validar hierarquia de componentes e as semelhanças visuais, através da comparação da geração do modelo e a interface original. Para aplicação industrial foi conduzida entrevistas com profissionais da área.	91.1 média de precisão entre os componentes. Hierarquia de componentes mais próxima da verdadeira que os outros modelos comparados. As aplicações geradas têm grande similaridade com as aplicações originais. Os profissionais entrevistados deram avaliações positivas mas seriam necessárias modificações para uso em indústria
(Wimmer, 2021)	*Ad-hoc* com base na avaliação informal por humano	Criado experimento para avaliar o modelo, e também um questionário	Dos 129 componentes desenhados pelos participantes 121 foram corretamente

			identificados, e 126 foram corretamente posicionados. Todos os participantes conseguiram usar a aplicação sem problemas, avaliando com média de 7.1 (nota de 1 à 10) a usabilidade.
(Sun, 2020)	Utilizadas as medidas acurácia, média de *recall* e precisão.	Usados 10% do conjunto de dados para teste.	Acurácia de 96.97%, precisão e *recall* de 86.4%.

Tabela 14 - Técnicas utilizadas para avaliação

Abdelhamid et al. (2020), Baura et al. (2022) e Sun et al. (2020) utilizaram medidas comuns na avaliação de modelos de detecção de objetos tais como acurácia, precisão e *recall*. Abdelhamid et al. (2020) utiliza estes valores para comparar seu modelo com outros presentes na literatura.

Baulé et al. (2021) utilizou um questionário, para o qual foram gerados wireframes a partir de *sketches* randomicamente selecionados do conjunto de teste, e questionado à voluntários o quão fiel o wireframe gerado é comparado com o *sketch* associado. Os participantes também foram solicitados a gerar seus próprios *sketches* e utilizar a aplicação.

Moran et al. (2020) teve o processo mais elaborado, utilizou a média de precisão para avaliar a efetividade da geração. Utilizou verificação visual para validar a hierarquia da gerada e a similaridade visual entre o gerado e o original. Também utilizou a validação de especialistas para concluir a usabilidade na indústria.

Wimmer et al. (2021) tinha como foco a geração em tempo real, então foi criado um ambiente onde foi colocado um quadro branco com uma câmera capturando a imagem e enviando para um computador para gerar e apresentar a interface. Os participantes tiveram que criar 3 interfaces pré-definidas e uma a sua escolha.

3.4 Discussão

Estendendo a análise do estado da arte levantado nos anos anteriores (revisão de Baulé et al. 2021), podemos observar que ainda existe relevância na geração automática de interfaces a partir de *sketches*.

Nenhum dos trabalhos analisados tem como objetivo criar *wireframes* em penpot a partir de *sketches,* como também para nenhuma ferramenta de *design* gráfico, tendo como foco apenas aplicações web ou *mobile*.

Podemos ver também a evolução para extração de mais características dos *sketches*, como por exemplo Wimmer et al. 2021 que estende seu trabalho para alinhar os

componente nos *wireframes* finais e Barua et al. (2022) que utiliza símbolos para criar navegação entre as telas identificadas e também faz a identificação de texto para criar *wireframes* mais completos.

Nos trabalhos analisados existe uma predominância de CNNs na etapa de processamento de imagens, e o uso de conjuntos proprietários de dados para seu treinamento e avaliação.

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

- A omissão de artigos relevantes é um risco que foi mitigado ao fazer uma string de busca com diversos sinônimos e ao realizar a pesquisa em várias bases de dados.
- As ameaças na seleção e extração de dados foram mitigadas através da definição de critérios de inclusão/exclusão e de qualidade, que foram revisados e seguidos rigorosamente.
- A probabilidade de publicações de resultados positivos é maior que a de resultados negativos, o que sempre apresenta um possível viés para mapeamento sistemático. Este viés não representa uma ameaça grave porque os artigos foram utilizados na avaliação do estado da arte e tendo assim pouca influência sobre este mapeamento sistemático.
- Pelo fato de um único pesquisador, o autor deste artigo, ter participado nesta revisão sistemática, a imparcialidade na aplicação dos critérios de inclusão e exclusão pode ter sido afetada.

4. Modelo Sketch2Penpot

Neste capítulo é apresentado o modelo de geração automática de *wireframes* para Penpot a partir de *sketches* feitos em papel.

4.1 Análise de requisitos

Neste trabalho é desenvolvido um modelo que tenha como entrada *sketches* de interfaces de usuário, e automaticamente gera os *wireframes* que a representam em um projeto Penpot. Este modelo é dividido em duas etapas. Tendo como entrada os *sketches*, a primeira etapa realiza a detecção dos diferentes componentes e suas posições. Nesta etapa são utilizadas estratégias de visão computacional usando deep learning para a detecção de objetos, com base na pesquisa feita foi escolhido o modelo YOLOv8 por apresentar melhores resultados em termos de desempenho.

A entrada são fotografias de *sketches* desenhados à mão, cada uma representando uma tela do aplicativo. Cada elemento a ser identificado requer um padrão para que possa ser identificado. A Tabela 15 exemplifica como cada elemento que pode ser identificado pela rede deve ser desenhado nos *sketches* que serão utilizados como entrada. Outros elementos ou outras maneiras de os representar não são considerados e podem resultar em falsos positivos.

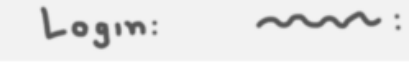
Label	
Button	
Image	
Textbox	
ListPicker	
Switch	
CheckBox	
Slider	
Map	

Tabela 15 - Representação no *sketch* de cada elemento que pode ser identificado

Como saída é gerado um json com uma lista de elementos, cada um contendo as seguintes informações:

- Categoria
- Identificador da categoria
- Coordenadas do ponto central
- Tamanho
- Caminho para o arquivo da foto

A segunda etapa tem como entrada os resultados da detecção de objetos de cada componente de interface encontrada (saída da etapa 1), e transforma essas informações para um projeto de Penpot, onde cada componente identificado estará presente. Assim a saída desta etapa é um arquivo de exportação de penpot (.svg + json).

		Representação dos componentes em memória		
Imagem do	Etapa 1 -		Etapa 2 -	*Wireframe*

Sketch	Detecção dos componentes		geração do *wireframe*	(arquivo .svg + json)

Tabela 16 - Etapas de geração de *wireframes* a partir de arquivos de exportação Penpot

Requisitos funcionais

RF1. Detectar elementos de *design* de UI em *sketches*

Na primeira etapa devem ser detectados componentes em fotos de *sketches* desenhados à mão. Utilizando o template de Mitchell (1997), esta tarefa é definida como um programa de computador que aprende quando sua experiência E, resulta em uma melhoria de desempenho medida com P, em relação a uma classe de tarefas T. Assim, a tarefa (T) é identificar os diferentes componentes de interface de usuário nas fotografias dos *sketches* de entrada, a experiência (E) é o conjunto de *sketches* com os devidos componentes identificados (Conforme especificados na Tabela 15) e o desempenho (P) é a mAP50 que avalia a precisão média entre as classes identificadas com confiança acima de 50%, para este trabalho um valor de 75% é considerado o mínimo satisfatório. O resultado da detecção de objetos deve ser representado em forma de um json com a lista de elementos identificados.

RF 2. Geração automática de projeto penpot a partir da detecção de objetos

Na segunda etapa, utilizando a representação em json com a lista de elementos identificados (categoria e coordenadas) (saída da etapa anterior) como entrada, deve ser gerado um projeto penpot (arquivo svg + json) com os mesmos componentes representados.

Requisitos não funcionais

RnF 1. *Performance*	O tempo para a geração do projeto penpot não deve levar mais do que 5 min para 5 telas.
RnF 2. Disponibilização	Deve ser implementado como um serviço web instalado no servidor no SETIC/UFSC.

4.2 Desenvolvimento do modelo de detecção de componentes nos *sketches*

4.2.1 Preparação do conjunto de dados

Para o treinamento do modelo de deteção de objetos é necessário um conjunto de fotos de *sketches* pareado com um arquivo identificando a posição e tipo de cada componente representado na imagem (Figura 20). Foi utilizado o conjunto criado por Baulé et al. (2021) como base, do qual foram usados 480 *sketches* de interfaces de usuário.

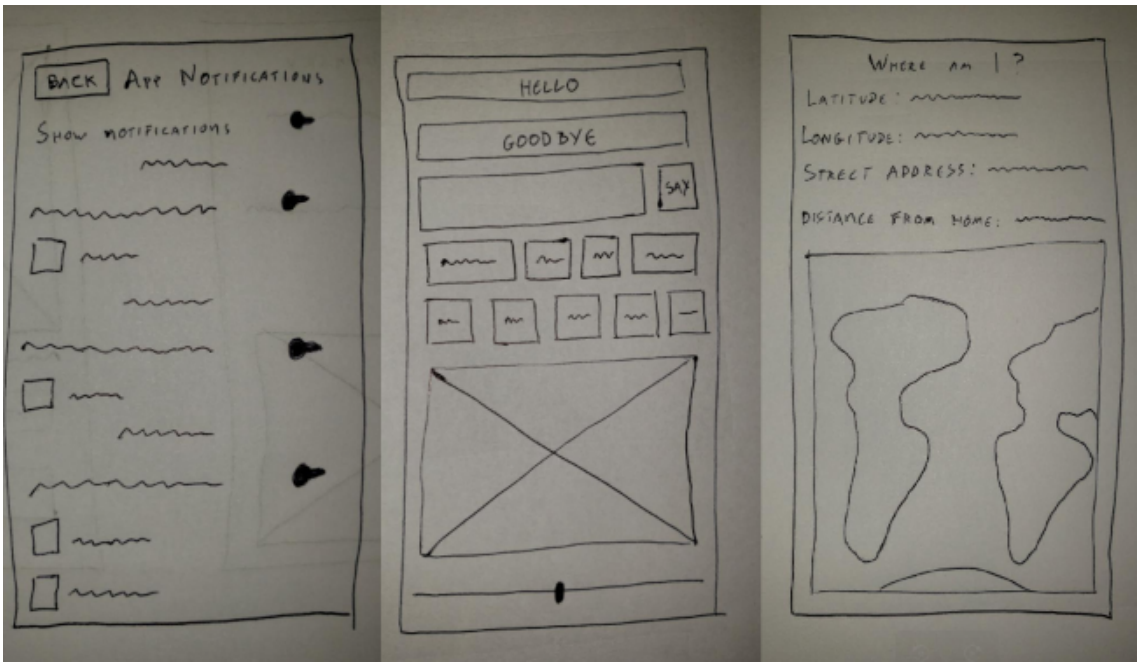


Figura 20 - Exemplos de *sketches* separados do conjunto de Baulé et al. (2020)

Depois do processo de *labelling* foram adicionados 40 novos *sketches* contendo maior densidade dos componentes menos presentes no conjunto de base. A Tabela 14 lista os componentes a serem detectados com exemplos de *sketches*.

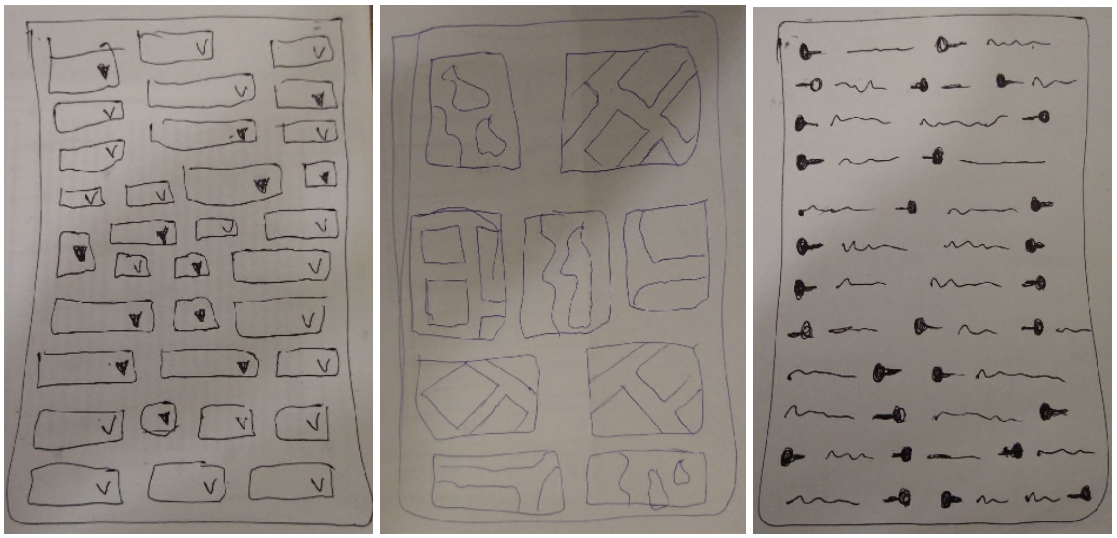


Figura 21 - Exemplos dos novos *sketches* contendo menos presentes

O tamanho total do conjunto de imagem são 520 *sketches* sendo 480 de interfaces de apps e 40 focados em aumentar os exemplos de elementos menos representados.

Labeling das imagens

Para cada *sketch* considerado foi realizado o *labeling* manual dos diferentes componentes presentes. Para este processo foi utilizada a plataforma Roboflow (2020), devido a sua coleção de ferramentas que facilitam a organização e criação das *labels,* como também a exportação para diferentes modelos de visão computacional (Figura 22).

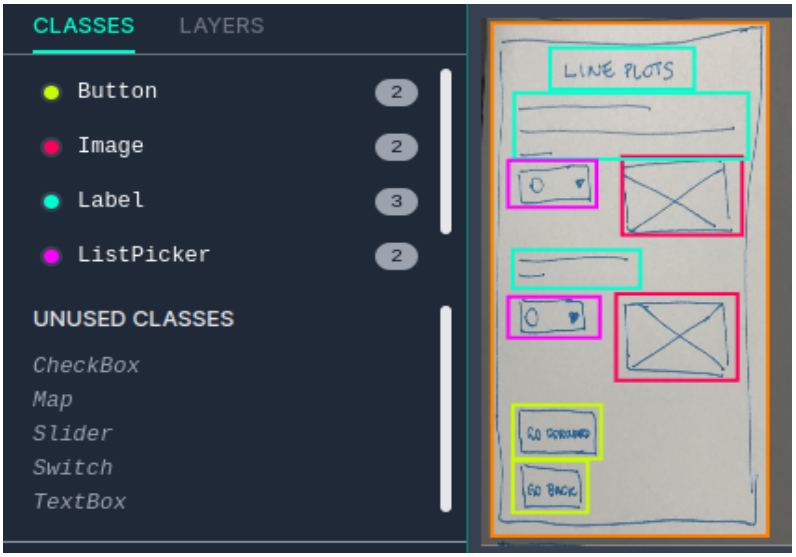


Figura 22- Exemplo de *labeling* no Roboflow

Como o conjunto base considerou aplicativos criados para a plataforma App Inventor, a frequência de aparição de certos componentes é maior que a de outros (Tabela 17).

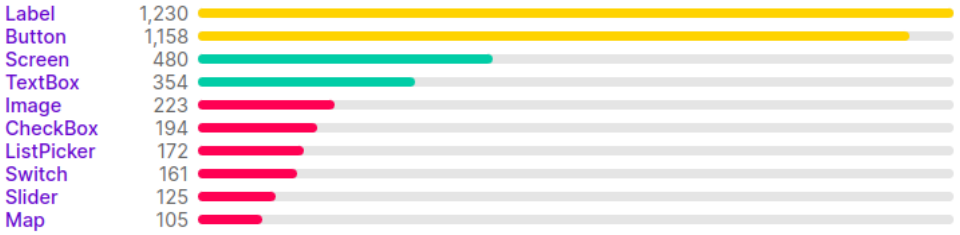


Tabela 17 - Frequência de componentes no conjunto base

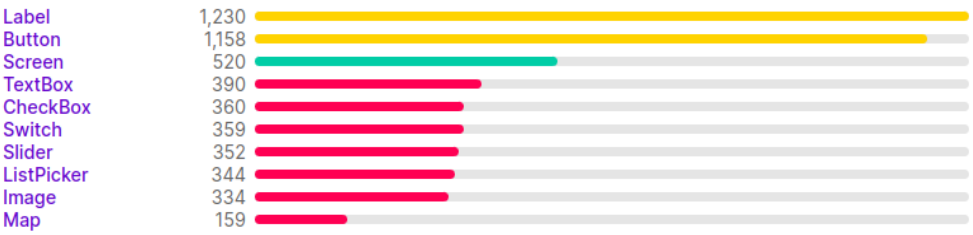


Tabela 18 - Frequência de componentes no conjunto final

Para aumentar o tamanho do conjunto de treinamento foram utilizadas ferramentas de *data augmentation* da plataforma roboflow, criando assim fotos borradas, com *pixels* de barulho e também imagens rotacionadas no eixo central vertical (Figura 21)(Tabela 19).

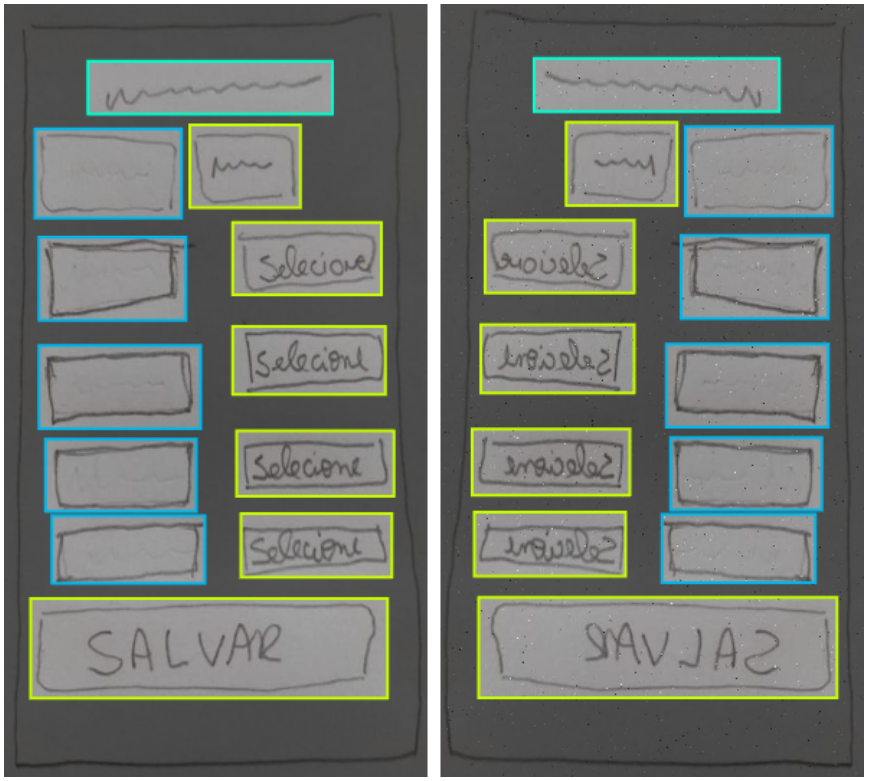


Figura 23 - Exemplo de imagem com *data augmentation*

	Conjunto de imagens			
	Total	**Baulé et al. 2021**	**Novos sketches**	**Link do conjunto de imagens**
sem *data augmentation*	520	480	40	https://universe.roboflow.com/ufsc-lcdfw/sketch2penpot
com *data augmentation*	1272	--	--	--

Tabela 19 - Resumo do conjunto de dados usado

O conjunto final foi dividido em 1.128 imagens de treinamento, 96 para validação e as últimas 48 para teste.

Além dos elementos de interface a serem detectados, foram criadas *labels* para a tela do aplicativo que será utilizada para redimensionar os elementos detectados.

4.2.2 Treinamento de modelo YOLO8

O treinamento YOLO8 (REF) foi realizado na plataforma Google Colab, tendo como base o disponibilizado pela Ultralythics. Vários tamanhos de modelos de YOLO8 são disponibilizados pela Ultralythics, o que causa variação na precisão e no tempo de execução (Tabela 18). Para obtermos um equilíbrio entre o tempo de execução e a precisão do sistema foi utilizado o modelo YOLOv8m.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Tabela 20 - Modelos disponibilizados pela Ultralythics (Ultralythics 2023).

A rede foi treinada por 300 épocas, com tamanho de lote 32 tamanho de imagem 800 *pixels*. Após 190 épocas treinadas, o modelo não obteve resultados melhores durante 50 épocas e o treinamento foi encerrado. Obtivemos um valor de mPA50 de 87.5%, precisão de 91%, e taxa de *recall* de 81.1% (Tabela 19). Na Tabela 20 é apresentada a matriz de confusão final das classes consideradas.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	96	775	0.91	0.811	0.875	0.797
Button	96	193	0.87	0.901	0.935	0.869
CheckBox	96	49	1	0.733	0.913	0.809
Image	96	48	0.975	0.826	0.911	0.864
Label	96	222	0.833	0.739	0.84	0.623
ListPicker	96	24	0.888	0.661	0.793	0.759
Map	96	19	0.909	0.789	0.804	0.786
Screen	96	96	0.995	1	0.995	0.975
Slider	96	33	0.98	0.879	0.942	0.841
Switch	96	22	0.847	0.773	0.802	0.679
TextBox	96	69	0.801	0.814	0.817	0.761

Tabela 21 - Resultados de desempenho finais do treinamento

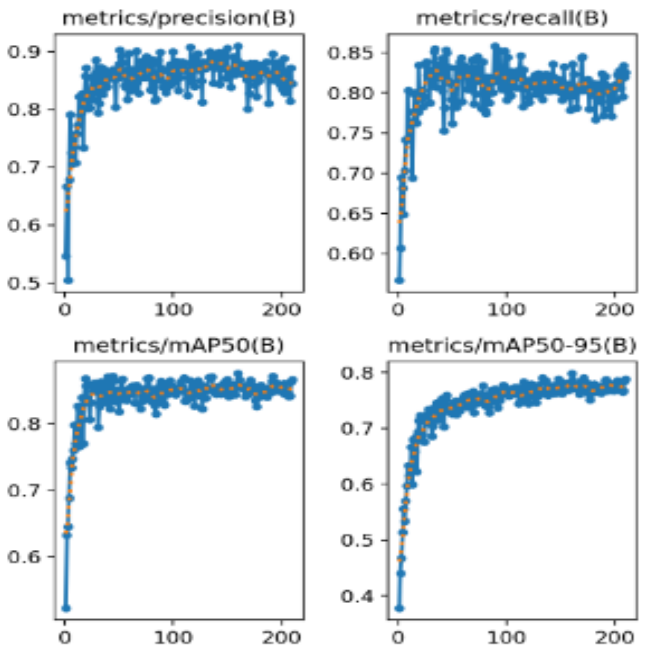


Figura 24 - Evolução do desempenho avaliada durante o treinamento

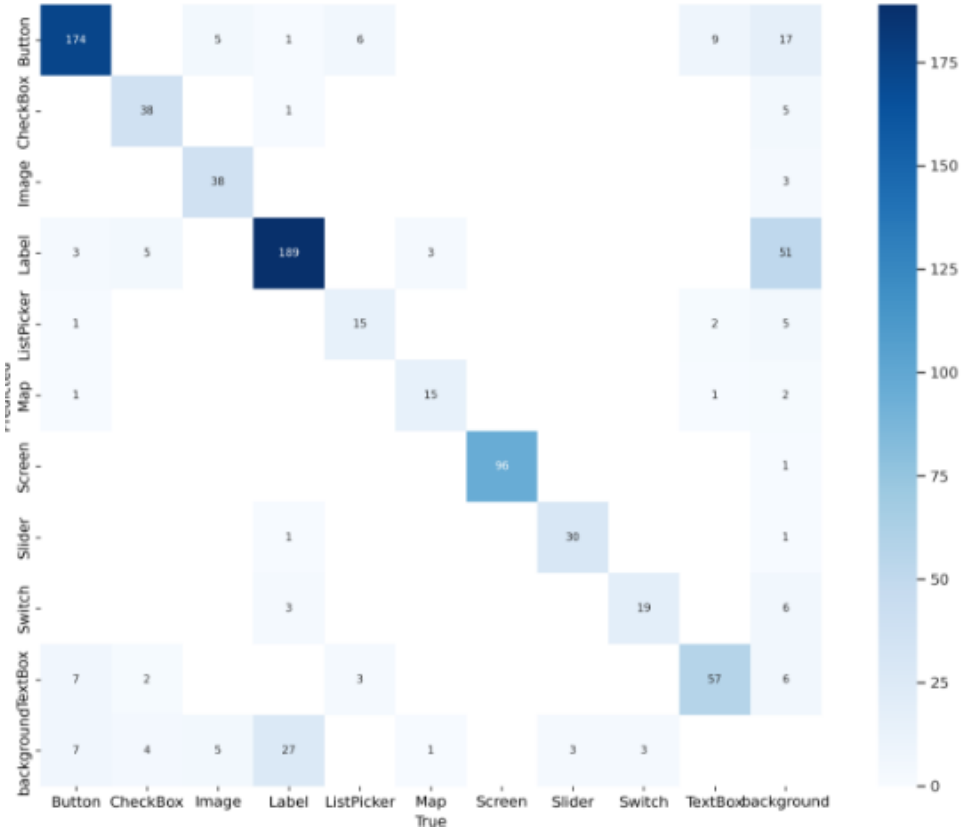


Tabela 22 - Matriz de confusão no resultado do treinamento .

Ao avaliar as métricas de avaliação do modelo (Tabela 21) e a matriz de confusão (Tabela 22) observa-se que a classe *Label* obteve o maior número de falsos positivos e menor valor de AP, isso possivelmente acontece porque elementos como *Sliders* e

CheckBoxes tem *labels* como parte de seus componentes. Outro motivo é que os Labels podem ter múltiplas linhas que durante o processo de *labeling* foram marcados como um componente único, mas a rede identifica como múltiplas *Labels.*

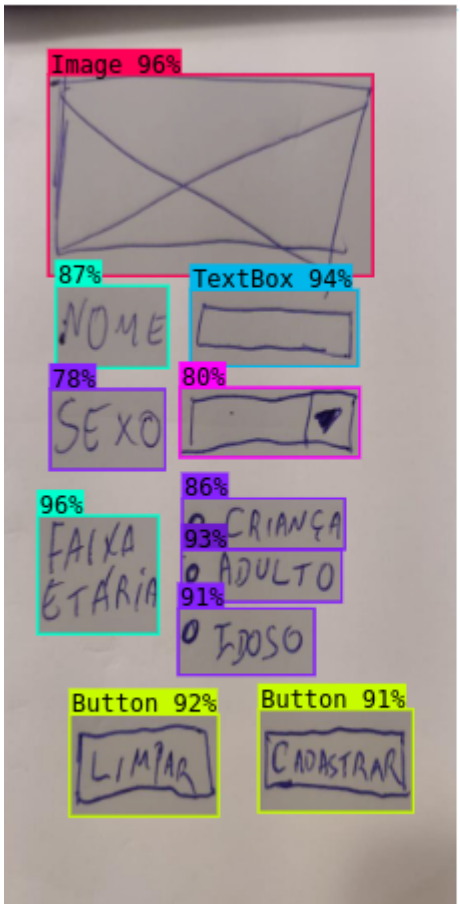
A rede treinada foi disponibilizada na plataforma Roboflow: https://universe.roboflow.com/ufsc-lcdfw/sketch2penpot/model/2.

4.2.3 Resultado da detecção de objetos

Depois de treinada a rede pode ser exportada em diversos formatos. Foi utilizado o formato apropriado para PyTorch. A partir do modelo exportado é possível fazer novas predições. O modelo foi disponibilizado na plataforma Roboflow que permite facilmente a importação em outros sistemas.

Por meio da biblioteca Roboflow, a rede neural foi importada em um programa Python, que carrega as fotos a partir de um diretório e executa a detecção dos elementos de interface presentes em cada .png no diretório.

O resultado da detecção de elementos para cada tela é um objeto do qual pode ser extraído um json com um vetor de elementos detectados (Figura 25).




```
"predictions": [
  {
    "x": 197.5,
    "y": 750,
    "width": 147,
    "height": 144,
    "confidence": 0.964,
    "class": "Label",
    "class_id": 3
  },
  {
    "x": 334.5,
    "y": 263.5,
    "width": 395,
    "height": 245,
    "confidence": 0.96,
    "class": "Image",
    "class_id": 2
  },
  {
    "x": 412,
    "y": 449.5,
    "width": 204,
    "height": 93,
    "confidence": 0.944,
    "class": "TextBox",
    "class_id": 8
  },
  {
    "x": 396.5,
    "y": 751,
    "width": 195,
    "height": 64,
    "confidence": 0.929,
    "class": "CheckBox",
    "class_id": 1
  },
```


Figura 25 - Exemplo de json de saída após a detecção dos elementos

Cada elemento no .json é descrito como um dicionário com sua classe (*class, class_id*), sua posição central (x, y), seu tamanho (*width, height*) e o grau de confiança da detecção da rede.

Também é salvo um .png da foto original marcada com as *bounding boxes* que foram identificadas pela detecção como também seu grau de confiança, com objetivo de ajudar na validação do projeto (Figura 25).

4.3 Geração do projeto penpot

Após a fase de detecção, é percorrido o conjunto de elementos detectados, e com suas características são atualizados os valores dos componentes Penpot base associados ao elemento de interface detectado.

Para cada tipo de elemento de interface a ser identificado foi criado um componente Penpot, com base no conjunto *Material Design 3* disponibilizado pela Google. Estes componentes foram exportados como a combinação de arquivos .svg e .json. Destes arquivos foram alterados os ids dos elementos e componentes como também as coordenadas de posição e valor tamanho para chaves que depois serão alteradas pelos valores finais (Tabela 23).

Chave final	**Valor original**	**Descrição**
button_background_id	e5695384-3158-8064-8003-68a95e95dd63	Identificador do retângulo de fundo
button_label_id	e5695384-3158-8064-8003-68a95e95dd65	Identificador do texto
Button_group_id	e5695384-3158-8064-8003-68b98cc216fd	Identificador do componente
button_background_component_id	e5695384-3158-8064-8003-68b98cc259f1	Identificador do retângulo de fundo no componente
button_label_component_id	e5695384-3158-8064-8003-68b98cc259f2	Identificador do texto no componente
background_height, background_width	128, 40	Tamanho do retângulo de fundo
background_position_x, background_position_y	0, 0	Posição do retângulo de fundo
text_position_x, text_position_y	43, 28	Posição do texto
label_position_x, label_position_y	43, 11	Posição da caixa de texto

Tabela 23 - Valores trocados por chaves no .svg base do elemento Botão

Para cada tela é utilizado o valor do elemento *Screen* para calcular o desvio dos elementos dentro da foto do *sketch* em relação a tela desejada (Figura 26).Também é calculada uma escala para normalizar as telas em um tamanho de 720 por 1280 (Figura 27).

```
def get_offset( element ):
    offset = {"x" :0, "y": 0}
    offset["x"] = element["x"] - ( element["width"] / 2 )
    offset["y"] = element["y"] - ( element["height"] / 2 )
    return offset
```

Figura 26 - Função que calcula desvio

```
def get_scale( element ):
    scale = {"x" :1, "y": 1}
    scale["x"] = screen_size["width"] / element["width"]
    scale["y"] = screen_size["height"] / element["height"]
    return scale
```

Figura 27 - Função que calcula escala

Para cada elemento do vetor de saída da etapa anterior, é utilizado a classe para carregar o elemento básico como uma *string*. Com a posição central e o tamanho do elemento é calculada a posição do vértice superior da esquerda do componente Penpot (Figura 28), que é utilizado para atualizar a *string* carregada.

```
element_offset = get_offset(element)
element_x = (element_offset["x"] - offset["x"]) * scale["x"]
element_y = (element_offset["y"] - offset["y"]) * scale["y"]
element_width = element["width"] * scale["x"]
element_height = element["height"] * scale["y"]
```

Figura 28 - Cálculo do vértice do topo esquerdo de elemento

Também é atualizado o tamanho do componente. Cada elemento de interface é composto por componentes básicos diferentes que têm que ser atualizados com valores diferentes, então uma função específica para cada foi criada (Figura 29).

```
def move_button( button_str, positionx, positiony, height, width ):
    button_str = move(button_str, "background", str(positionx), str(positiony))
    positionx_str = positionx + ( width / 3 ) # calcula offset do texto
    positiony_str = positiony + ( height / 3) # calcula offset do texto
    button_str = move_text( button_str, positionx_str, positiony_str )
    return button_str

def resize_button( button_str , height, width ):
    button_str = resize(button_str, "background", str(height), str(width))
    return button_str
```

Figura 29 - Funções que posicionam e escalam botões.

Utilizando a biblioteca Python *svgutils*, é criado um objeto svg a partir da *string* atualizada, que posteriormente é adicionado a um objeto svg que representa a tela do aplicativo.

Para cada classe de elemento identificado é adicionado o componente apropriado ao *components.svg*, as características dos componentes não precisam ser alteradas pois só descreve a associação entre os elementos básicos que o compõem.

Depois de todos os elementos do vetor terem sido processados, os .svg gerados são salvos na pasta apropriada, que é comprimida com o *manifest.json* para gerar .zip com o projeto Penpot final (Figura 30).

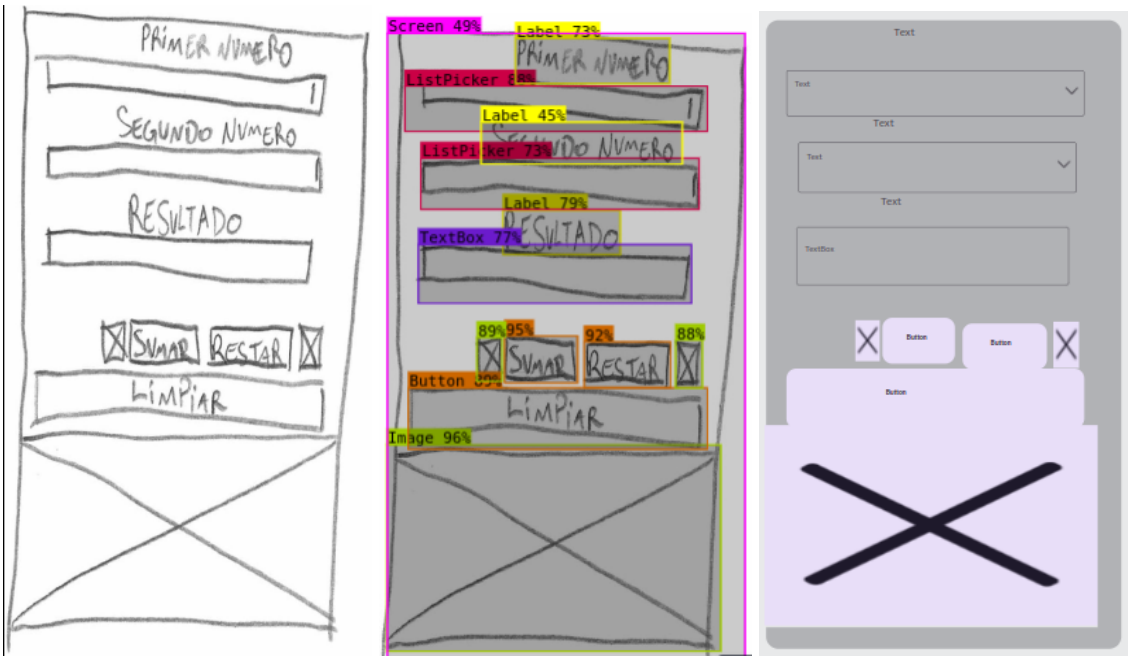


Figura 30- Exemplo do processo de geração

4.4 Aplicativo web

O aplicativo web foi desenvolvido a partir do aplicativo web Sketch2aia, disponibilizado por Baulé (2020), pois ambos os trabalhos têm o objetivo de gerar *wireframes* a partir de fotos de sketches de interface de usuário.

A aplicação permite que seja feito o *upload* de múltiplas imagens (Figura 31), cada imagem deve representar apenas uma tela de aplicativo, e todas farão parte do mesmo projeto Penpot no final da geração. Ao fazer o *upload* é gerado um código alfanumérico de 5 dígitos, que representará o projeto, e é criado um diretório com mesmo nome onde são salvas as imagens, e servirá como base para a conversão.

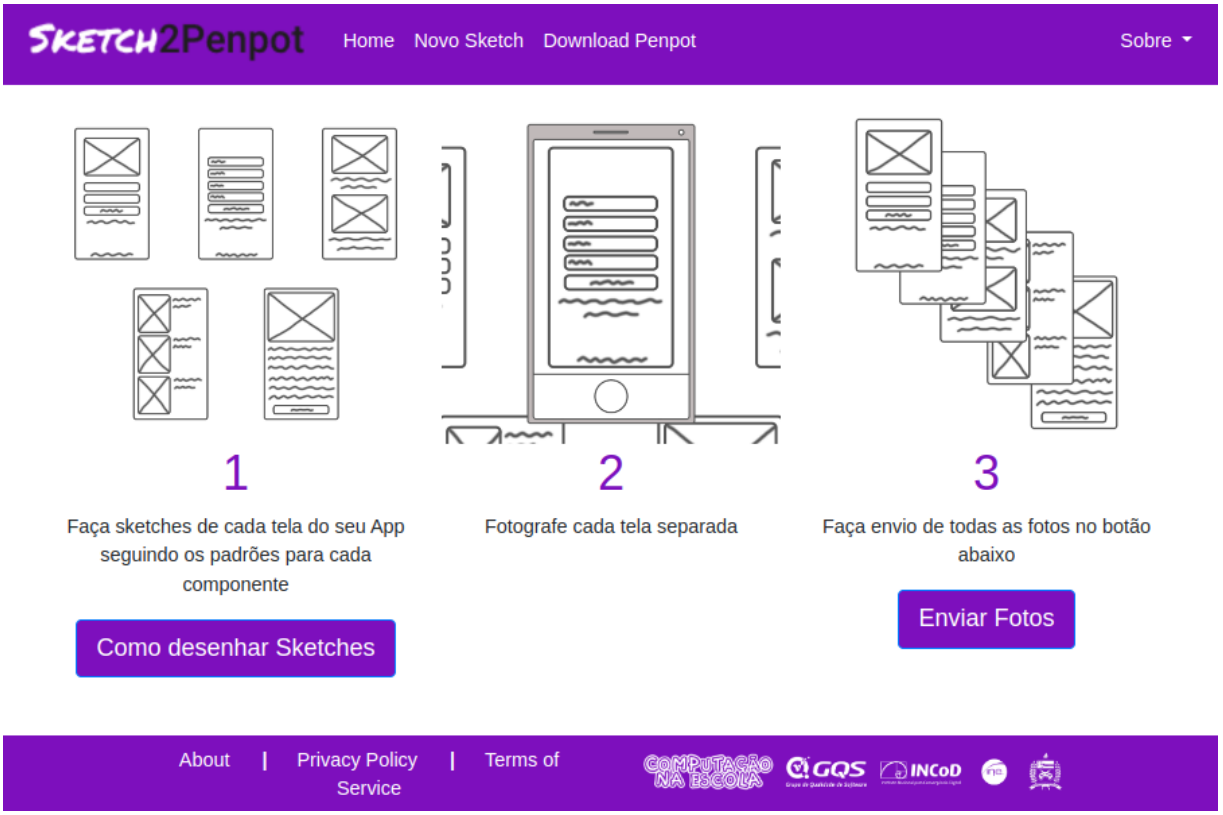


Figura 31 - Tela de upload da ferramenta

Feito o *upload* das imagens para o servidor, é passado o código do projeto para o módulo Python criado para geração do projeto final, que percorre o diretório nomeado com o mesmo código e processa cada .png ou .jpg presente. Depois de gerado o projeto final, são apresentadas as imagens que foram carregadas com as *bounding boxes* dos elementos identificados pela rede neural e é possível fazer o *download* do .zip contendo o projeto Penpot em formato pelos .svg de cada tela e o de componentes como também o *manifest.json* com informações do projeto (Figura 32).

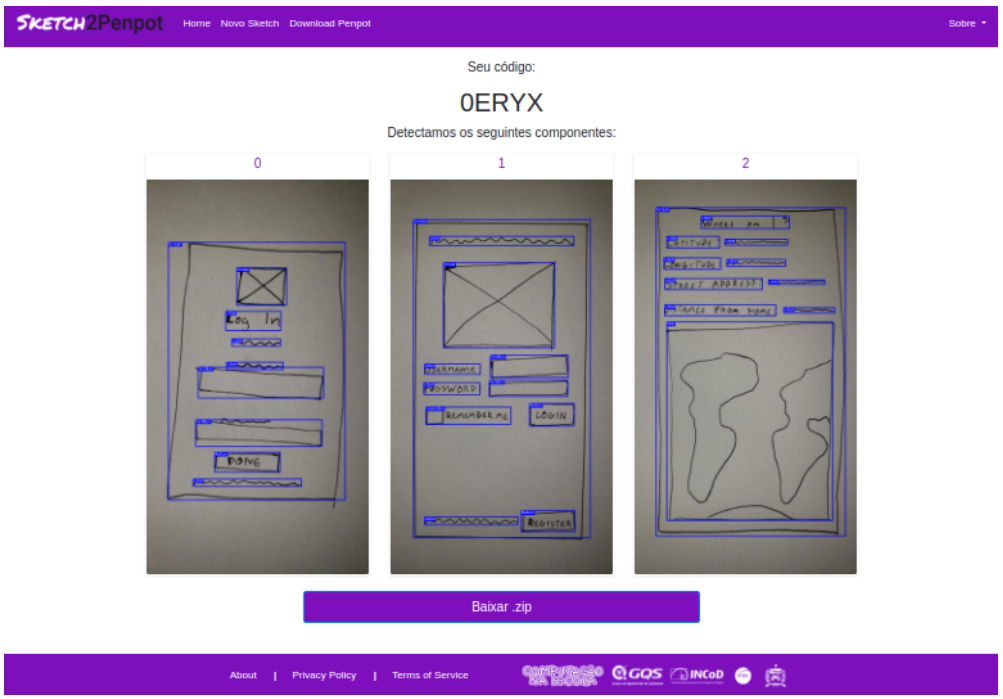


Figura 32 - Tela de visualização/*download* do projeto

Utilizando o código do projeto também é possível fazer o *download* de projetos que foram processados anteriormente (Figura 33).

Figura 33 - Tela de *download* de projetos previamente processados

Com o .zip baixado é possível fazer o *upload* na plataforma Penpot, o projeto é nomeado com o mesmo código usado no *download* e contém uma página para cada tela que foi carregada no sistema.

O código para aplicação foi disponibilizado online: https://codigos.ufsc.br/gqs/Sketch2Penpot.

5. Conclusão

Este trabalho tinha como objetivo principal a criação de um modelo que possibilitasse a geração automática de projetos penpot a partir de *sketch* desenhados à mão. Foi feito o levantamento e análise do estado da arte, identificando a não existência de um modelo que tivesse como objetivo a criação de *wireframes* para a ferramenta proposta. Observamos um desempenho satisfatório, tanto no tempo de execução como também na qualidade dos *wireframes* gerados. Um problema do modelo apresentado é a impossibilidade de gerar *switches* a direita do texto, pois o *switch* e o texto são considerados um elemento só ao fazer a detecção independente do lado em que são dispostos. Textos escritos que acabem em “o” ou outras letras redondas podem ser categorizados como *checkboxes* pelo menos motivo apresentado anteriormente.

Para trabalhos futuros, sugere-se a melhoria na geração dos *wireframes*, como por exemplo criar métodos para alinhamento dos componentes identificados. Outro ponto de interesse seria ampliar o conjunto de dados de treinamento, visando melhorar a detecção feita pela rede proposta como também acrescentar novos componentes a serem identificados. Com a intenção de melhor validação da aplicação e de sua utilidade também é proposto um teste com usuários finais.

Referências

Abdelhamid, A. A.; Alotaibi S. R.; Mousa A.; **Deep learning-based prototyping of android GUI from hand-drawn mockups**. In: The Institution of Engineering and Technology, 2020

Barua S.S.; Zulkarnain I. M.; A. R.; Alam G.R.; Uddin Z.; **Sketch2FullStack: Generating Skeleton Code of Full Stack Website and Application from Sketch using Deep Learning and Computer Vision**, arXiv, 2022

Baulé, D.;**Desenvolvimento de um Modelo de Geração Automática de Wireframes no App Inventor a partir de Sketches usando Deep Learning,** ine.tcc, Brasil, 2020

Baulé, D.; Gresse von Wangenheim, C.; von Wangenheim, A. ; Hauck, J. C. R.; Vargas Júnior, E. C.; **Automatic code generation from sketches of mobile applications in end-user development using Deep Learning.** arXiv, 2021

Baulé, D.; Gresse von Wangenheim, C.; von Wangenheim, A.; Hauck, J. C. R.; Vargas Júnior, E. C.; **Using Deep Learning to Support the User Interface Design of Mobile Applications with App Inventor.** IHC '21: Proceedings of the XX Brazilian Symposium on Human Factors in Computing Systems, Brasil, 2021

Baulé, D.; Gresse von Wangenheim, C.; von Wangenheim, A; Hauck, J. C. R.; Vargas Júnior, E. C.; **Recent Progress in Automated Code Generation from GUI Images Using Machine Learning Techniques.** Journal of Universal Computer Science, vol. 26, n 9, 2020

Bochkovskiy, A.; Wang, C.; Liao, H. M.; **YOLOv4: Optimal Speed and Accuracy of Object Detection**, 2020 Disponível em <https://arxiv.org/abs/2004.10934> Acesso em abril 2023

Boesch G.; **YOLOv7: The Most Powerful Object Detection Algorithm (2023 Guide).** 2023 Disponivel em <https://viso.ai/deep-learning/yolov7-guide/> Acesso em março 2023

Brahmbhatt P.; **The YOLO Object Detection.** 2019 Disponivel em <https://medium.com/@prashant.brahmbhatt32/the-yolo-object-detection-c195994b53aa> Acesso em Março 2023

Córdova-Esparza, D. M.; Terven, J. **A COMPREHENSIVE REVIEW OF YOLO: FROM YOLOV1 TO YOLOV8 AND BEYOND.** 2023 Disponível em: <https://arxiv.org/pdf/2304.00501.pdf>

Goodfellow, I.; Bengio, Y.; Courville, A.; **Deep Learning**. Cambridge, MA: MIT Press, 2016.

Gutta, S.; **Object Detection Algorithm - YOLO v5 Architecture.** 2021 Disponível em <https://medium.com/analytics-vidhya/object-detection-algorithm-yolo-v5-architecture-89e0a35472ef> Acesso em março 2023

Hui, J.; **Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3**, 2018 Disponível em:

<https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088> Acesso em abril 2023

LeCun, Y.; Bengio, Y.; Hinton G.; **Deep learning**, 2015 Disponível em <https://www.nature.com/articles/nature14539> Acesso em abril 2023
Jain, V.; Agrawal, P.; Banga, S.; Kapoor, R.; Gulyani, S.; **Sketch2Code: Transformation of Sketches to UI in Real-time Using Deep Neural Network.** arXiv, 2019

Kaleidos, **Features,** 2023 Disponível em: <https://penpot.app/features>. Acesso em abril 2023

Kaleidos, **Technical Guide**, 2023 Disponível em <https://help.penpot.app/technical-guide/developer/data-model/ > Acesso em abril 2023

Kukil; Rath, S.; **YOLOv7 Object Detection Paper Explanation & Inference.** 2022 Disponivel em <https://learnopencv.com/yolov7-object-detection-paper-explanation-and-inference/> Acesso em março 2023

MIT App Inventor. **About Us**, 2023 Disponível em: <http://appinventor.mit.edu/explore/about-us.html>. Acesso em abril 2023

Moran, K.; Bernal-Cardenas, C.; Curcio, M.; Bonett, R.; Poshyvanyk, D.; **Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps.** IEEE Transactions on Software Engineering, 2020

Pathmind, **A Beginner's Guide to Convolutional Neural Networks (CNNs).**
Disponível em: <https://wiki.pathmind.com/convolutional-network>. Acesso em:
abril de 2023

Roboflow, **Roboflow Documentation,** 2020 Disponível em: <https://docs.roboflow.com>. Acesso em Outubro 2023

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

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

Subramanyam, V. S.; **IOU (Intersection over Union).** 2021 Disponivel em <https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef> Acesso em março 2023

Sun, X.; Li, T.; Xu, J.; **UI Components Recognition System Based On Image Understanding.** IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Macau, China, 2020

Ultralytics; **Comprehensive Guide to Ultralytics YOLOv5.** 2023 Disponível em <https://docs.ultralytics.com/yolov5/> acesso em março 2023

Ultralytics; **Object Detection YOLOv8.** 2023 Disponível em <https://docs.ultralytics.com/tasks/detect/> acesso em outubro 2023

Ultralytics; **Quickstart YOLOv8.** 2023 Disponível em <https://docs.ultralytics.com/tasks/quickstart/> acesso em outubro 2023

Wang, C.; Bochkovskiy, A.; Liao H. M.; **YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.** 2022 Disponível em <https://arxiv.org/pdf/2207.02696.pdf> Acesso em março 2023

Wimmer, C.; Untertrifaller, A.; Grechenig, T.; **SketchingInterfaces: A Tool for Automatically Generating High-Fidelity User Interface Mockups from Hand-Drawn.** OzCHI '20: Proceedings of the 32nd Australian Conference on Human-Computer Interaction, Sydney, Australia, 2021

Apêndice A

Código disponibilizado em: https://codigos.ufsc.br/gqs/Sketch2Penpot

Para sua execução basta seguir as instruções no README.md

Apêndice B

Desenvolvimento de Sketch2Penpot

Joaquim B. Belo

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

joaquim.boschini@grad.ufsc.br

Abstract. *Converting user interface sketches into wireframes in a visual design application is a necessity both in application development and in the teaching of visual design in schools. As this task is resource-intensive, this work presents an approach that automates this process. This application utilizes deep learning techniques to detect components and their positions in photos provided by users, creating a Penpot project with the wireframes of the detected elements. The network model achieves an average accuracy of 87.5% in classifying user interface elements. This process automation has been made available through a web tool.*

Resumo. *Transformar sketches de interfaces de usuários em wireframes numa aplicação de design visual é uma necessidade tanto no desenvolvimento de aplicações como no ensino de design visual nas escolas. Como esta tarefa é muito custosa, este trabalho apresenta uma abordagem que automatiza este processo. Esta aplicação usa técnicas de deep learning para detectar os componentes e suas posições em fotos passadas pelos usuários, e cria um projeto Penpot com os wireframes dos elementos detectados. O modelo da rede atinge uma precisão média de classificação de elementos de interface de usuário de 87,5%. Esta automação do processo foi disponibilizada em uma ferramenta web.*

1. Introdução

Com o contínuo aumento do uso aplicativos para dispositivos móveis, a necessidade de processos ágeis de desenvolvimento e iteração também aumenta. Uma das principais áreas em que/para qual existe um grande investimento e retorno é no desenvolvimento das interfaces de usuário.

O desenvolvimento de interfaces normalmente envolve um processo de prototipação em que são elaboradas *sketches* que são representações simples dos elementos gráficos, tipicamente feitas em papel e canetas. A partir dos sketches são criados *wireframes*, estruturas de *design* sem detalhes visuais como por exemplo cores, que definem a estrutura base da interface (Abdelhamid, 2020). E ao final é feito o *design* visual adicionando os detalhes de cor, tipografia, ícones e imagens tornando os wireframes em uma representação mais fiel do produto.

O desenvolvimento de *wireframes* e do *design* visual é tipicamente feito usando ferramentas de *design* gráfico. Um exemplo é a Penpot é uma ferramenta de *design*, de código aberto que permite a prototipação de diversos produtos, dentre eles interfaces de usuário. Tem como objetivo aproximar desenvolvedores e *designers* de maneira a tornar o processo criativo mais escalável, sendo uma alternativa gratuita a outras ferramentas, como por exemplo o Figma. Penpot utiliza elementos simples, como linhas e círculos,

que podem ser agrupados de forma a criar elementos mais complexos. Projetos podem ser importados e exportados por meio de arquivos .json e .svg ou no formato proprietário .penpot.

Pela natureza custosa e iterativa do processo, a automação se torna de grande valor. Uma das partes que pode ser automatizada é a geração de *wireframes* a partir de sketches e possibilitando assim uma continuação do processo de *design* adicionando elementos do *design* visual como cor e fontes. Nos últimos anos foram desenvolvidas várias soluções para a transformação de *sketches* para *wireframes* em HTML, *Sketch*, React, Android Studio entre outros, utilizando diversas técnicas de visão computacional clássica, redes neurais convolucionais ou alguma combinação destas técnicas (Robinson, 2019). Porém, atualmente ainda não existe uma ferramenta que permita a criação automática de *wireframes* no Penpot a partir de *sketches* de papel.

Neste contexto, este artigo apresenta a ferramenta Sketch2Penpot, uma solução que cria automaticamente wireframes de interfaces de usuário na ferramenta Penpot a partir de *sketches*, utilizando técnicas de *deep learning* para fazer a detecção de elementos de interface que é usada para criar o projeto final.

2. Trabalhos relacionados

Com a importância do *design* de interface e a possível redução do esforço pela geração automática do código da interface de usuário (UI), atualmente existem esforços de pesquisa com esse objetivo, mas nenhum com a intenção de criação de wireframes em uma plataforma para o refinamento do *design* visual.

Dos trabalhos existentes o foco é na geração de *wireframes* para aplicações web e mobile. A maioria das abordagens tem como entrada imagens de *sketches* de interface de usuários, e alguns utilizam screenshots de interfaces já existentes como por exemplo Moran (2020) e Sun (2020). Os conjuntos de elementos considerados variam, mas o foco é sempre nos elementos mais presentes em aplicações já existentes.

Mesmo com o aumento da pesquisa em geração automática de interfaces, ainda não existe a disponibilidade de um grande conjunto de dados para treinamento, portanto os trabalhos relacionados que pretendem identificar elementos em *sketches*, fazem o seu desenvolvimento e anotação.

Em termos de redes neurais utilizadas para a detecção dos elementos de interface de usuário, podemos ver uma clara preferência por redes neurais convolucionais (CNN). Trabalhos que pretendem ir além da geração da interface, como por exemplo Barua (2022), utilizam outra rede para identificar textos, de maneira a criar *wireframes* mais completos.

A validação dos modelos normalmente tem foco na detecção da rede neural utilizada e usa valores de média de precisão (mAP) e tempo de execução. Alguns dos trabalhos analisados, como por exemplo Baulé (2020) e Wimmer (2021), utilizaram formulários de avaliação e teste com usuários finais para validar não só a detecção dos elementos mas também os *wireframes* gerados no final.

3. Metodologia de Pesquisa

Analisando os resultados de trabalhos relacionados a diferentes tipos de UI (*user interface*), desenvolvemos uma ferramenta Web para a geração automática de projeto Penpot com base em *sketches*, seguindo uma abordagem multi-métodos.

Fundamentação teórica. Durante esta fase foram analisadas a teoria sobre design de interfaces de usuário de apps App Inventor, a síntese de conceitos de representação de projetos no Penpot e síntese de conceitos básicos sobre *deep learning* especificamente detecção de elementos de interface.

Estado da arte. Nesta etapa é realizado um mapeamento sistemático da literatura para identificar e analisar modelos de geração automatizada de *wireframes* de interfaces atualmente publicados.

Desenvolvimento do modelo semântico por detecção de objetos. 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. Foi utilizado o conjunto de treinamento disponibilizado por Baule (2020) e criados novos *sketches* para complementar os elementos menos presentes. Foram utilizadas métricas comuns na literatura (mAP) para a sua avaliação.

Desenvolvimento do código de Penpot. Foi desenvolvido um módulo de *software* que utilizando o resultado da detecção de objetos, cria um projeto Penpot com os elementos detectados e suas posições. Este módulo foi disponibilizado em uma ferramenta web que aceita o *upload* de imagens de *sketches* e gera o projeto final, que pode ser facilmente importado na plataforma Penpot.

4. Sketch2Penpot

Neste trabalho é desenvolvido um modelo que tenha como entrada *sketches* de interfaces de usuário, e automaticamente gera os wireframes que a representam em um projeto Penpot. Este modelo é dividido em duas etapas. Tendo como entrada os *sketches*, a primeira etapa realiza a detecção dos diferentes componentes e suas posições, que são passadas para a segunda etapa, que com base em elementos básicos criados previamente, cria novos elementos e faz as alterações necessárias para criar o *wireframe* esperado.

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.

Preparação dos Dados. Foi utilizado o conjunto de dados disponibilizado por Baule como base, seu trabalho tinha como foco aplicações em App Inventor, o que causou seu

conjunto de dados ter uma grande diferença em quando cada elemento está representado. Para contrariar este problema foram criados novos *sketches* com a intenção de aumentar o número dos componentes menos representados no conjunto base, no final foram considerados 520 *sketches*.

Foi usada a plataforma Roboflow para o processo de anotação e foram utilizadas técnicas de *data augmentation* para aumentar o conjunto de treinamento, resultando em um conjunto final de 1128 imagens para treino, 96 para validação e 48 para teste

Aprendizado do Modelo. Foi utilizado modelo YOLOv8 por apresentar grandes melhorias na família YOLO como também estar entre os melhores resultados quando comparada com outras redes na tarefa de detecção de objetos. O modelo foi desenvolvido e mantido pela Ultralytics, e é disponibilizado em diversos tamanhos, que variam o tempo e a precisão da detecção. Os modelos são pré-treinados e validados utilizando o conjunto COCO, e podem ser importados para diversas plataformas.

O treinamento foi iniciado considerando 300 épocas, com tamanho de lote 32 tamanho de imagem 800 pixels. Após 190 épocas treinadas, o modelo não obteve resultados melhores durante 50 épocas e o treinamento foi encerrado.

Avaliação de Desempenho. Foram usadas métricas comuns na literatura para a avaliação do modelo. O valor de previsão média (*Average Precision*) é definido como a precisão de detecção média em diferentes *recalls* e geralmente é avaliado de uma maneira específica da classe de objeto. O AP médio (mAP) é calculado sobre todas as classes de objetos e é usado o valor de 50% como limiar de confiança para o valor de precisão ser considerado como correto durante a avaliação.

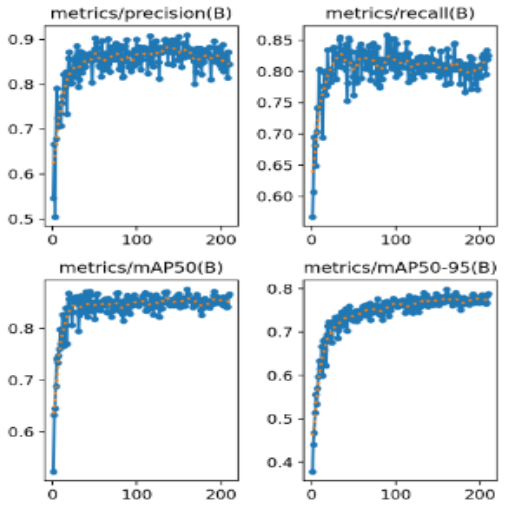


Figura 1. Evolução das métricas analisadas durante o treinamento

A Figura 1 mostra a evolução do desempenho do modelo durante seu treinamento. Após 140 épocas foram obtidos os melhores resultados, um valor de mAP50 de 87.5%, precisão de 91% e taxa de *recall* de 81.1%. A Figura 2 mostra os valores de desempenho finais para cada classe identificada pelo modelo.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	96	775	0.91	0.811	0.875	0.797
Button	96	193	0.87	0.901	0.935	0.869
CheckBox	96	49	1	0.733	0.913	0.809
Image	96	48	0.975	0.826	0.911	0.864
Label	96	222	0.833	0.739	0.84	0.623
ListPicker	96	24	0.888	0.661	0.793	0.759
Map	96	19	0.909	0.789	0.804	0.786
Screen	96	96	0.995	1	0.995	0.975
Slider	96	33	0.98	0.879	0.942	0.841
Switch	96	22	0.847	0.773	0.802	0.679
TextBox	96	69	0.801	0.814	0.817	0.761

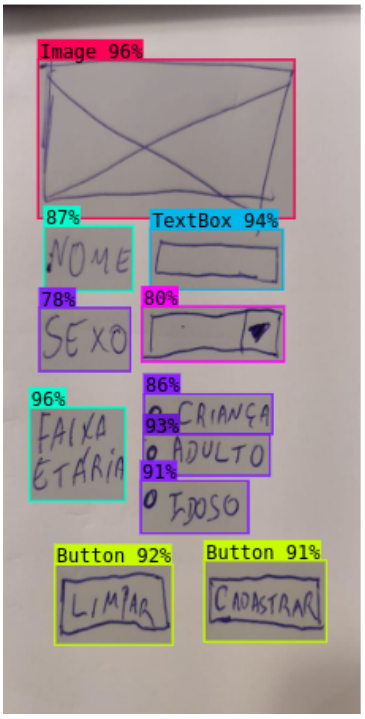
Figura 2. Valores de desempenho finais.

Ao avaliar as métricas de avaliação do modelo podemos observar que o elemento *ListPicker* obteve o menor resultado, possivelmente isto acontece porque é uma das classes menos representada no conjunto de treino, este problema também afetou outras como *Map* e *Switch*. O mesmo a classe *Label* ser uma das mais representadas, obteve um resultado menor que a média, isso provavelmente aconteceu porque *labels* com múltiplas linhas foram classificadas como um *label*, mas foi identificada como múltiplas *labels* durante o processo de detecção, outro possível motivo é componentes como *TextBox* e *Switch* terem como parte de seus elementos *labels*, o que pode ter causado falsos positivos.

O modelo final foi disponibilizado online na plataforma Roboflow: https://app.roboflow.com/ufsc-lcdfw/sketch2penpot/deploy/2

4.2. Gerando Código de Wireframes

A saída do processo de detecção de objetos é um vetor com os elementos identificados, descritos como um dicionário com sua classe, posição central e tamanho da *bounding box* e o grau de confiança da detecção da rede.




```
"predictions": [
  {
    "x": 197.5,
    "y": 750,
    "width": 147,
    "height": 144,
    "confidence": 0.964,
    "class": "Label",
    "class_id": 3
  },
  {
    "x": 334.5,
    "y": 263.5,
    "width": 395,
    "height": 245,
    "confidence": 0.96,
    "class": "Image",
    "class_id": 2
  },
  {
    "x": 412,
    "y": 449.5,
    "width": 204,
    "height": 93,
    "confidence": 0.944,
    "class": "TextBox",
    "class_id": 8
  },
  {
    "x": 396.5,
    "y": 751,
    "width": 195,
    "height": 64,
    "confidence": 0.929,
    "class": "CheckBox",
    "class_id": 1
  },
```


Figura 3. Exemplo da saída do processo de detecção de objetos

Para cada tipo de elemento de interface a ser identificado foi criado um componente Penpot, com base no conjunto *Material Design* 3 disponibilizado pela Google. Estes componentes foram exportados como a combinação de arquivos .svg e .json. Destes arquivos foram alterados os ids dos elementos e componentes como também as coordenadas de posição e valor tamanho para chaves que depois serão alteradas pelos valores finais.

Para cada tela é utilizado o valor do elemento *Screen* para calcular o desvio dos elementos dentro da foto do *sketch* em relação a tela desejada.Também é calculada uma escala para normalizar as telas em um tamanho de 720 por 1280.

Para cada elemento do vetor de saída da etapa anterior, é utilizado a classe para carregar o elemento básico como uma *string*. Com a posição central e o tamanho do elemento é calculada a posição do vértice superior da esquerda do componente Penpot, que é utilizado para atualizar a string carregada. Utilizando a biblioteca Python *svgutils*, é criado um objeto svg a partir da string atualizada, que posteriormente é adicionado a um objeto svg que representa a tela do aplicativo.

Depois de todos os elementos do vetor terem sido processados, os .svg gerados são salvos na pasta apropriada, que é comprimida com o *manifest.json* para gerar .zip com o projeto Penpot final.

4.3. Ferramenta Web

O aplicativo web foi desenvolvido a partir do aplicativo web Sketch2aia, disponibilizado por Baulé (2020). A aplicação permite que seja feito o upload de múltiplas imagens, cada imagem deve representar apenas uma tela de aplicativo, e todas farão parte do mesmo projeto Penpot no final da geração.

Feito o *upload* das imagens para o servidor, é passado o código do projeto para o módulo Python criado para geração do projeto final, que percorre o diretório nomeado com o mesmo código e processa cada .png ou .jpg presente. Depois de gerado o projeto final, são apresentadas as imagens que foram carregadas com as *bounding boxes* dos elementos identificados pela rede neural e é possível fazer o *download* do .zip contendo o projeto Penpot.

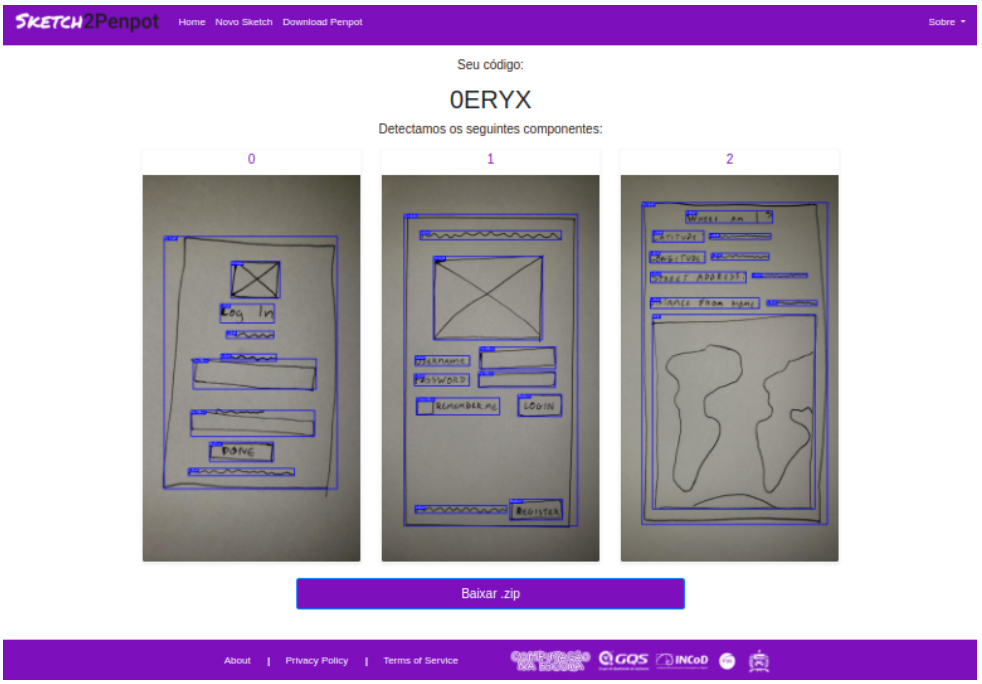


Figura 3. Tela de visualização/*download* do projeto

Codigo da ferramenta disponivel: https://codigos.ufsc.br/gqs/Sketch2Penpot

5. Conclusões

Este trabalho tinha como objetivo principal a criação de um modelo que possibilitasse a geração automática de projetos penpot a partir de *sketch* desenhados à mão. Foi feito o levantamento e análise do estado da arte, identificando a não existência de um modelo que tivesse como objetivo a criação de *wireframes* para a ferramenta proposta. Observamos um desempenho satisfatório, tanto no tempo de execução como também na qualidade dos *wireframes* gerados. Um problema do modelo apresentado é a impossibilidade de gerar *switches* a direita do texto, pois o *switch* e o texto são considerados um elemento só ao fazer a detecção independente do lado em que são dispostos.

Para trabalhos futuros, sugere-se a melhoria na geração dos *wireframes*, como por exemplo criar métodos para alinhamento dos componentes identificados. Outro ponto de interesse seria ampliar o conjunto de dados de treinamento, visando melhorar a detecção feita pela rede proposta como também acrescentar novos componentes a serem identificados. Com a intenção de melhor validação da aplicação e de sua utilidade é proposto um teste com usuários finais.

Referências

Abdelhamid, A. A.; Alotaibi S. R.; Mousa A. (2020) “Deep learning-based prototyping of android GUI from hand-drawn mockups.” In: The Institution of Engineering and Technology

Barua S.S.; Zulkarnain I. M.; A. R.; Alam G.R.; Uddin Z. (2022) “Sketch2FullStack: Generating Skeleton Code of Full Stack Website and Application from Sketch using Deep Learning and Computer Vision”, arXiv

Baulé, D. (2020) “Desenvolvimento de um Modelo de Geração Automática de Wireframes no App Inventor a partir de Sketches usando Deep Learning”, ine.tcc, Brasil

Moran, K.; Bernal-Cardenas, C.; Curcio, M.; Bonett, R.; Poshyvanyk, D. (2020) “Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps.” IEEE Transactions on Software Engineering

Robinson, A. (2019) “Sketch2code: Generating a website from a paper mockup.”, University of Bristol, UK

Sun, X.; Li, T.; Xu, J. (2020) “UI Components Recognition System Based On Image Understanding.” IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Macau, China

Wimmer, C.; Untertrifaller, A.; Grechenig, T.; SketchingInterfaces (2021) “A Tool for Automatically Generating High-Fidelity User Interface Mockups from Hand-Drawn.” OzCHI '20: Proceedings of the 32nd Australian Conference on Human-Computer Interaction, Sydney, Australia