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

ODIN - Uma Ferramenta Visual para o Desenvolvimento de Modelos de Detecção de Objetos com *Deep Learning* no Ensino Superior

João Pedro Santana

Florianópolis

2022

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística

ODIN - Uma Ferramenta Visual para o Desenvolvimento de Modelos de Detecção de Objetos com *Deep Learning* no Ensino Superior

Trabalho de Conclusão de Curso de Graduação em Sistemas de Informações, 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: João Pedro Santana

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

Florianópolis
2022

João Pedro Santana

ODIN - Uma Ferramenta Visual para o Desenvolvimento de Modelos de Detecção de Objetos com *Deep Learning* no Ensino Superior

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

Florianópolis, 20 de março de 2022

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

Prof. Dr. Jean Carlo Rossa Hauck
Coorientador
Universidade Federal de Santa Catarina

Prof. Dr. Mateus Grellert da Silva
Avaliador
Universidade Federal de Santa Catarina

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Nos últimos anos, projetos na área de *Machine Learning (ML) e especialmente de Deep Learning (DL)* estão presentes no dia a dia, em carros autônomos, antivírus e até no reconhecimento de padrões de imagens. Assim, o *Deep Learning* representa uma área de conhecimento importante na ciência da computação voltada a diversas tarefas como classificação de imagens ou detecção de objetos. Porém, sua complexidade pode ser uma barreira para os iniciantes, especialmente quando se utilizam ferramentas, arquiteturas e bibliotecas de contexto profissional baseadas em linguagens textuais de programação. Assim, um primeiro contato com essa tecnologia pode ser facilitado pelo uso de ambientes visuais para a criação de modelos de *DL que* permitam ensinar conceitos, sem necessitar competências relacionadas a programação, fornecendo uma interface gráfica intuitiva para construir seu modelo. Atualmente, já existem alguns ambientes desse tipo, porém, ainda não existe uma solução voltada ao ensino da tarefa de detecção de objetos inserida no contexto do ensino superior. Assim, o objetivo deste trabalho é desenvolver e avaliar, uma camada visual para suportar o processo de desenvolvimento de modelos de detecção de objetos no ensino superior. O trabalho inclui a síntese da fundamentação teórica, levantamento do estado da arte, e o desenvolvimento de uma camada visual para *Jupyter notebook* no *Google Colab*. Espera-se com isso ajudar os iniciantes a darem o primeiro passo para aprender conceitos básicos de *DL*, ao mesmo tempo preparando para uma fácil transição a ambientes convencionais usando linguagens de programação textuais.

Palavras Chaves: *Machine Learning*, *Deep Learning,* Ambiente visual, Detecção de objetos, Ensino superior, *Jupyter Notebook, Google Colab*

.

1.INTRODUÇÃO

1.1. CONTEXTUALIZAÇÃO

A Inteligência Artificial (IA) tem feito parte da imaginação por muitos anos nos filmes de ficção científica apresentando máquinas que fazem muito mais do que deveriam, como uma inteligência a qual poderia pensar como os humanos. Porém, esta ideia fica só em filmes e livros, pelo fato de ainda não ser possível a concretização desse sonho com as tecnologias existentes. Com este limite pressuposto das tecnologias atuais de nossas tecnologias criou-se um novo conceito "*Narrow Artificial intelligence*", tecnologias que são capazes de executar tarefas específicas tão bem quanto, ou até melhor, que humanos (Copeland, 2016). Com esta descoberta e mudança de percepção sobre a Inteligência Artificial criou-se o subcampo chamado de *Machine Learning (ML)* o qual tem o intuito de criar modelos que aprendam com experiências ou exemplos *(*Russel et al., 1995*). ML* é a prática de usar algoritmos para coletar dados, aprender com eles, e então fazer uma determinação ou predição sobre alguma coisa no mundo.

Deep learning (DL) é um subcampo de *Machine Learning*, o qual usa uma rede neural que tem como base o aprendizado por técnicas de *ML* (Hoover et al., 2019). O *DL* está cada vez mais presente, em produtos de consumo como celulares, computadores, câmeras e outros eletroeletrônicos, utilizado para combinar notícias, postagens ou produtos vinculados aos interesses dos usuários (Lecun et al., 2015). O *DL* permite que modelos computacionais compostos por várias camadas de processamento, aprendam representações de dados com múltiplos níveis de abstrações (Lecun et al., 2015). Estes métodos novos provenientes do *DL* melhoraram drasticamente o estado da arte no reconhecimento de fala, detecção de objetos e vários outros domínios em geral (Lecun et al., 2015).

Pela alta demanda por projetos na área da IA é estimada a criação de mais de 2 milhões de empregos na área em 2025 (Meulen et al., 2017), tornando-se uma das tecnologias que mais crescem no mundo da ciência da computação. Porém, esse crescimento até agora tem sido limitado pela falta de especialistas (Tamilselvam et

al., 2019). Com isso, instituições de ensino estão implementando em seus currículos, matérias com foco em IA, esperando que seus alunos consigam determinar quando é apropriado utilizar IA para resolver problemas e com isso aplicar o que foi aprendido (ACM, 2013). Porém, na prática se observam diversas dificuldades no ensino de *ML*, como por exemplo, a existência de um grande número de modelos, inúmeras bibliotecas (*Keras*, *Tensorflow*, *Pytorch, Modelzoos* e *Caffe*) e conhecimentos teóricos sobre *DL* (Tamilselvam et al., 2019). Assim, no primeiro contato, a aprendizagem sobre *ML* torna-se difícil pelo vasto número de *frameworks* e ferramentas que o programador iniciante precisa aprender a usar para então focar no aprendizado do *ML*. Para facilitar e aumentar a velocidade da aprendizagem de conceitos básicos de *ML* poderão ser utilizados ambientes visuais, os quais estão se tornando cada vez mais comuns na programação introdutória (Weintrop et al., 2017). Portanto, para popularizar o ML, é desejável reduzir o esforço cognitivo para que o usuário possa se concentrar na lógica para resolver o problema em questão (Knuth e Pardo, 1980). Para esse fim, foram introduzidas linguagens visuais que permitem que os usuários criem programas simplesmente arrastando e soltando um elemento visual em uma tela e, posteriormente, conectando esse elemento a outros elementos, em vez de especificá-los textualmente (Gresse von Wangenheim et al., 2021). Tais representações visuais podem assumir diversas formas, incluindo linguagens baseadas em blocos ou baseadas em fluxo (Gresse von Wangenheim et al., 2021). As linguagens visuais podem melhorar a capacidade de aprendizado de iniciantes, ajudando-os a evitar erros, favorecer o reconhecimento sobre a lembrança e fornecer conjuntos de instruções limitadas específicas de domínio, reduzindo a carga cognitiva (Gresse von Wangenheim et al., 2021). Essas vantagens levaram ao aumento da adoção em contextos de programação introdutória em diferentes estágios educacionais (Gresse von Wangenheim et al., 2021). As ferramentas visuais também devem ajudar a preparar os alunos para, posteriormente, aprenderem linguagens baseadas em texto em ambientes mais convencionais como Jupyter Notebooks usando Python (Armoni et al., 2015; Brown et al., 2016). Porém, mesmo surgindo cada vez mais ferramentas visuais para o desenvolvimento de modelos de ML, ainda observa-se a

falta de uma ferramenta visual para ensinar o desenvolvimento de modelos de detecção de objetos, dentro do contexto de um ambiente convencional como o Jupyter Notebook (https://jupyter.org/). Então, há a possibilidade de tornar-se uma solução para facilitar a aprendizagem inicial de conceitos de *Machine Learning* também na graduação.

1.2. OBJETIVOS

Objetivo Geral

O objetivo geral deste trabalho é desenvolver uma camada visual para suportar o processo de desenvolvimento de modelos de detecção de objetos em momentos iniciais do ensino de DL em cursos do ensino superior de computação. Essa camada visual deve suportar as principais etapas do processo de desenvolvimento de modelos de detecção de objetos, incluindo análise de requisitos, pré-processamento de dados, transferência de aprendizado, avaliação de desempenho, predição, e exportação do modelo. Visa-se a criação dessa camada visual para um ambiente profissional de desenvolvimento de modelos de *Machine Learning* (em *Jupyter Notebook no Google Colab* (https://colab.research.google.com)).

Objetivos Específicos

O1. Síntese da fundamentação teórica em relação ao conceito de ensino de *Machine Learning* no ensino superior, linguagens de programação visuais, ambientes de desenvolvimento e *frameworks* de *Machine Learning* e a tarefa de detecção de objetos em imagens usando *Machine Learning*.

O2. Análise do estado da arte em relação a ferramentas visuais para visualizar o processo de desenvolvimento de sistemas de *Machine Learning* para o ensino superior.

O3. Desenvolvimento da camada visual para suportar o processo de desenvolvimento de sistemas de *Machine Learning* no contexto do ensino superior.

1.3. METODOLOGIA

METODOLOGIA DE PESQUISA

Nessa pesquisa é usada uma abordagem multi-método. A metodologia de pesquisa utilizada neste trabalho é dividida nas seguintes etapas:

Etapa 1 – Fundamentação teórica

Etapa focada em estudar, analisar e sintetizar os conceitos principais e a teoria referente aos temas a serem abordados neste trabalho. Nesta etapa é apresentada a fundamentação teórica utilizando a metodologia de revisão narrativa (Cordeiro et al., 2007), e são realizadas as seguintes atividades:

A1.1 - Análise sobre ensino de *Machine Learning* no Ensino Superior.

A1.2 - Síntese dos conceitos de programação visual.

A1.3 - Síntese dos conceitos sobre *Machine Learning e* ambientes*/frameworks* existentes *e* tarefa de detecção de objetos em imagens usando *ML*.

Etapa 2 – Estado da arte

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

A2.1 – Definição do protocolo de revisão.

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

A2.3 – Extração e análise de informações relevantes.

Etapa 3 – Desenvolvimento

Nesta etapa é desenvolvido a camada visual seguindo um processo iterativo incremental de processo de desenvolvimento de software (Larman et al., 2003)

A3.1 - Análise de requisitos.

A3.2 - Modelagem da arquitetura do sistema.

A3.3 - Desenvolvimento de um modelo de *ML* para detecção de objetos.

A3.3 - Modelagem de baixo nível e implementação.

A3.4 - 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. O capítulo 4 apresenta a implementação da ferramenta visual ODIN e os seus respectivos resultados e funcionalidades. Para finalizar, o capítulo 5 apresenta a conclusão do presente trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos relevantes ao trabalho desenvolvido. Iniciando por uma análise teórica sobre o ensino de *Machine Learning* no ensino superior, seguido por uma síntese dos conceitos de programação em blocos, e por fim uma síntese dos conceitos sobre *Machine Learning.*

2.1. Deep Learning

O primeiro trabalho reconhecido como *Artificial Intelligence (*AI) foi realizado por Warren McCulloch e Walter Pitts (1943). Eles se basearam no conhecimento da fisiologia básica, função dos neurônios no cérebro e a teoria da computação de Turing (Russell et al., 1995). Com essas teorias criadas e aperfeiçoadas pelo tempo, abriu-se portas para criação de novos algoritmos, ideias e assim podendo chegar aos avanços da AI conhecidos nos tempos atuais, o qual impacta positivamente nossa sociedade.

AI abrange muitas áreas de uso em geral, como percepção e raciocínio lógico, para tarefas específicas, como jogar xadrez, provar teoremas matemáticos, escrever poesia e diagnosticar doenças (Russell et al., 1995). A AI é separada em duas dimensões principais entre os seres humanos a qual possui uma abordagem empírica e a racionalidade a qual envolve uma combinação de matemática e engenharia (Russell et al., 1995).

Machine Learning (*ML*) tem o intuito de montar modelos que aprendam com experiências ou exemplos *(*Russel et al., 1995*). ML* de maneira mais básica é a prática de usar algoritmos para coletar dados, aprender com eles, e então fazer uma determinação ou predição sobre algo no mundo (Copeland, 2016). *ML* permite que se aprenda constantemente com os dados, de modo a possivelmente prever mudanças futuras. Este conjunto de algoritmos e modelos tem sido usado em diversos mercados como forma de melhorar processos e ganhar aprofundamento em padrões e anomalias em dados (Hurwits e Kirsch, 2018).

O *Deep Learning (DL*) é um subcampo de *Machine Learning*, o qual usa uma rede neural que tem como base o aprendizado por técnicas de *ML* (Hoover et al., 2019). *Deep Learning* permite que modelos computacionais com múltiplas camadas

de processamento aprendam representações de dados com múltiplos níveis de abstração (Lecun et al., 2015). Estes métodos melhoraram drasticamente o estado da arte em reconhecimento de voz, reconhecimento visual de objetos, detecção de objetos e muitos outros domínios como pesquisa em drogas e genética (Lecun et al., 2015). A Figura 1 ilustra o relacionamento das diferentes disciplinas de *IA*, mostrando que *Deep Learning* é um tipo de *Machine Learning*, que é utilizado por várias, mas não todas, as abordagens de *IA*.

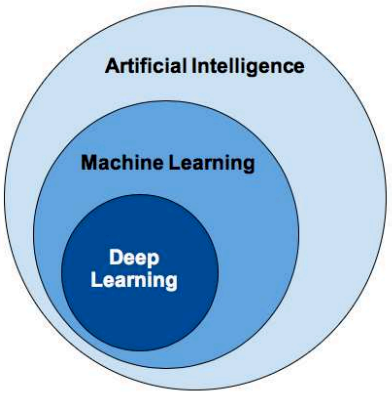


Figura 1 - Relacionamento entre subáreas de IA (Kenji, 2019)

Redes Neurais. A Rede Neural do ponto de vista biológico, trata-se de um modelo matemático inspirado em operações do cérebro humano *(*Russel et al., 1995*)*. Os elementos aritméticos simples da computação correspondem aos neurônios e a rede como um todo corresponde a uma rede neural, ou seja, uma coleção de neurônios interconectados *(*Russel et al., 1995*)*.

Uma rede neural é composta por vários nós, conectados por *links*. Cada link tem um peso numérico associado a ele *(*Russel et al., 1995*)*. Algumas das unidades estão conectadas ao ambiente externo e podem ser designadas como unidades de entrada ou saída, os pesos são modificados para tentar alinhar o comportamento da rede ao ambiente que fornece os dados brutos *(*Russel et al., 1995*)*. A Figura 1 mostra um exemplo da estrutura de um nó.

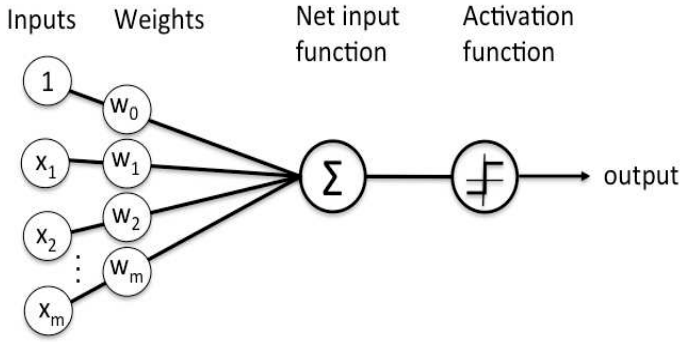


Figura 2 - Diagrama exemplo de um nó (Pathmind, 2019a)

As redes neurais foram projetadas para reconhecer padrões (Pathmind, 2019a). Elas interpretam os dados sensoriais através de um tipo de percepção da máquina, rotulando ou agrupando dados brutos (Pathmind, 2019a). Os padrões que eles reconhecem são numéricos, contidos em vetores, nos quais todos os dados do mundo real, sejam imagens, som, texto ou séries temporais, devem ser traduzidos. (Pathmind, 2019a). São geralmente divididas em camadas, no modelo mais básico de rede, temos uma camada de entrada de nós de fonte que se projeta sobre camadas ocultas intermediárias até uma camada de saída (Haykin, 2007).

Redes Neurais Profundas. Uma Rede Neural Profunda é a estrutura de rede utilizada em *Deep Learning*. Estas redes se distinguem por sua profundidade, ou seja, o número de camadas pelo qual o dado deve passar em um processo com múltiplas etapas de reconhecimento (Pathmind, 2019a). Em redes de *Deep Learning*, cada camada de nós treina em um conjunto distinto de características baseado na saída da camada anterior. Quanto mais se avança na rede neural, mais complexas as características reconhecidas pelos nós, devido a agregação e combinação de características da camada anterior (Pathmind, 2019a).

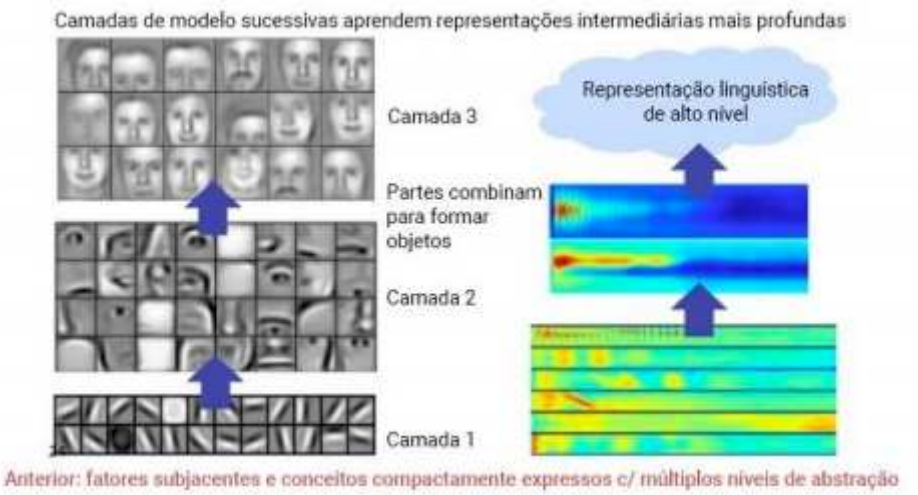


Figura 3 - Hierarquia de características (adaptado a partir do PATHMIND, 2019a)

Como demonstrado na Figura 3, isto é conhecido como hierarquia de características, e é uma hierarquia de complexidade e abstração incrementais, e faz com que redes de *Deep Learning* sejam capazes de lidar com grandes conjuntos de dados de alta dimensão com bilhões de parâmetros (Pathmind, 2019a). Assim, estas redes são capazes de descobrir estruturas latentes em dados não estruturados e não categorizados, o que caracteriza a grande maioria dos dados no mundo real (Pathmind, 2019a).

Estes dados também são conhecidos como mídia crua, como fotos, texto, vídeo e gravações de áudio (Pathmind, 2019a). Portanto, um dos problemas solucionados por *Deep Learning* é o processamento e agrupamento da mídia crua e não-categorizada do mundo, detectando similaridades e anomalias em dados que nenhum humano organizou em um banco de dados relacional ou até mesmo nomeou (Pathmind, 2019a)

Redes Neurais Convolucionais (*CNNs*). As Redes Neurais Convolucionais (*Convolutional Neural Networks*) são redes neurais primariamente utilizadas para classificação/agrupamento de imagens e reconhecimento de objetos. São capazes de reconhecer rostos, indivíduos, sinais de trânsito, animais e diversos outros aspectos de mídia visual (Pathmind, 2019c). A eficácia de redes neurais convolucionais em reconhecimento de imagem é um dos principais fatores para o reconhecimento recente da eficácia do *deep learning*. Hoje, CNNs são o centro de

grandes avanços na área de visão computacional, com uso em carros autônomos, robótica, drones, segurança, diagnósticos médicos e tratamento para deficientes visuais (Pathmind, 2019c). O modelo de redes neurais convolucionais engloba diversas arquiteturas, que variam em função da ativação dos nós, tipos de camadas etc.

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

Em uma *CNN*, geralmente são utilizadas diversas sequências de filtros diferentes para mapear as ocorrências de diferentes características da imagem (Pathmind, 2019c). Este tipo de rede neural enxerga imagens como volumes, ou seja, objetos tridimensionais. Isto ocorre devido a imagens digitais serem geralmente representadas utilizando uma codificação *Red-Blue-Green (RGB)*, com seus canais de cores sendo armazenados separadamente, e misturados para produzir o espectro de cores perceptível pelo olho humano. Cada canal de cor é representado por uma matriz de valores, que representam a intensidade do canal em questão em um determinado pixel (Pathmind, 2019c). A Figura 4 demonstra uma aproximação da representação de um canal de cor em uma imagem.

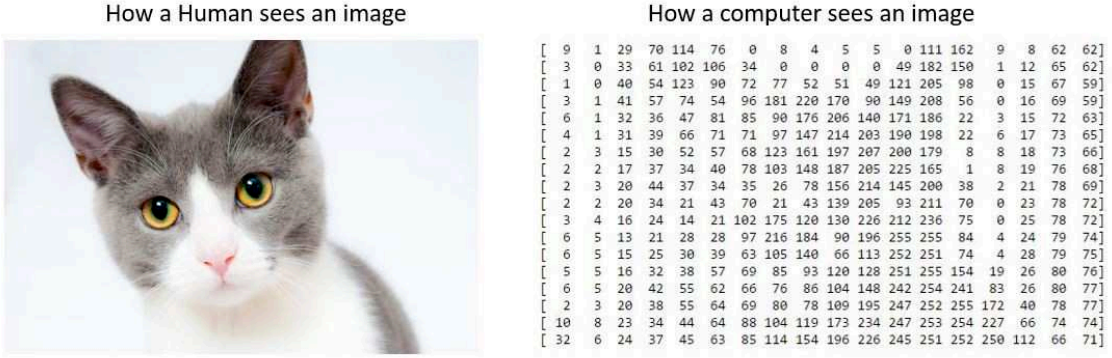


Figura 4 - Comparação de uma imagem com sua forma matricial (Karkare, 2019)

O filtro utilizado para convolução é uma pequena matriz de valores, que percorre a imagem um pixel por vez. O filtro (verde) desliza sobre a imagem de entrada (azul) um pixel de cada vez a partir do canto superior esquerdo. O filtro

multiplica seus próprios valores com os valores sobrepostos da imagem enquanto desliza sobre ela e adiciona todos eles até a saída de um único valor para cada sobreposição (Karkare, 2019). Como mostrado na Figura 5 a entrada e o filtro e a execução da operação de convolução na Figura 6.

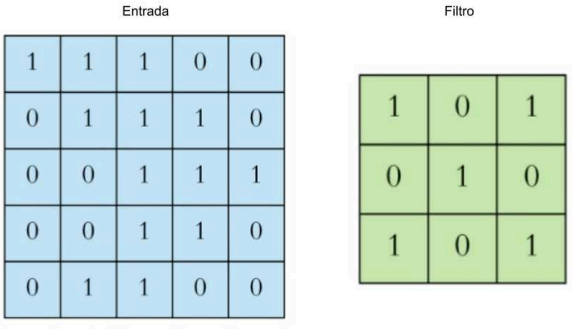


Figura 5 - Entrada e Filtro da operação de convolução

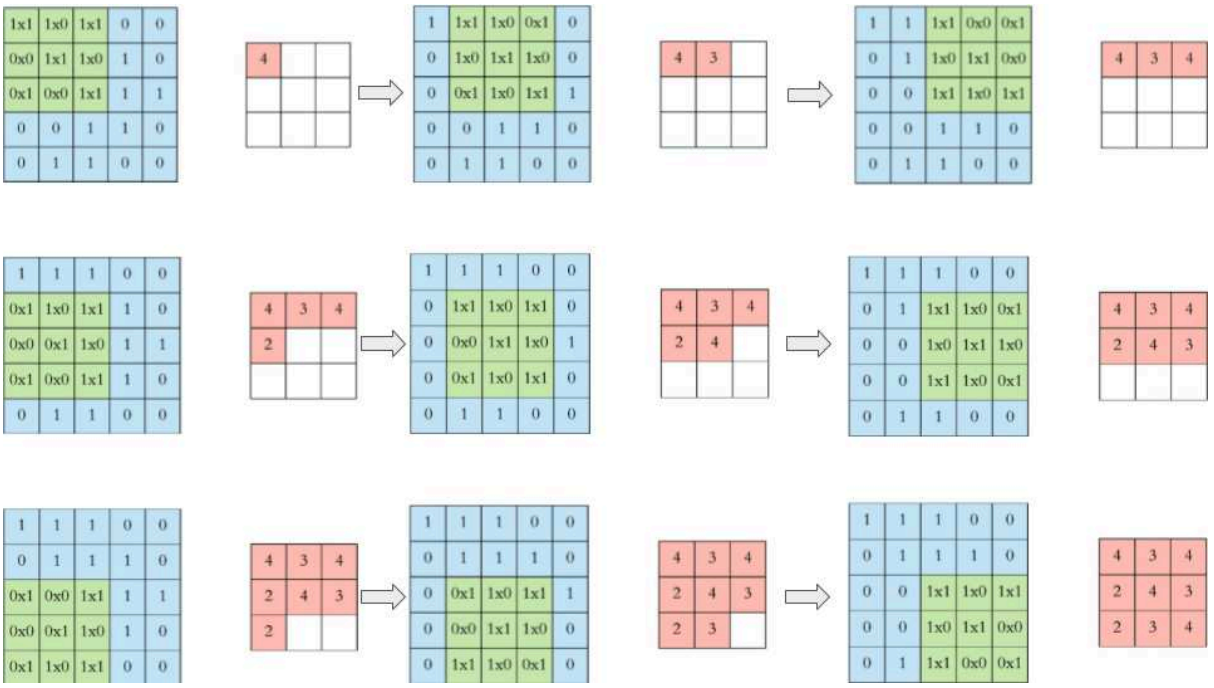


Figura 6 - Exemplo da execução da operação de convolução (adaptado a partir do (Karkare, 2019))

A aplicação de convolução permite realçar certas características da imagem, como a presença de linhas horizontais e verticais (Karkare, 2019) como mostrado na Figura 7.

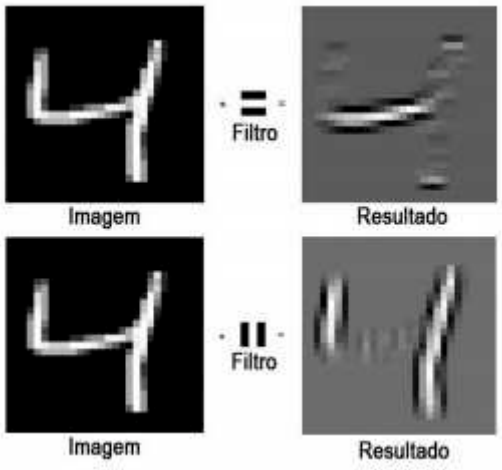


Figura 7 - linhas horizontais e verticais realçadas pela operação de convolução (adaptado a partir do (Karkare, 2019))

É comum adicionar em CNNs uma camada de *pooling*, também conhecido como *downsampling* ou *subsampling*. Esta camada tem como objetivo reduzir as dimensões dos dados, de modo a reduzir o número de parâmetros e computação na rede (Karkare, 2019). O tipo de pooling mais utilizado é o *max pooling*, onde se divide o dado em janelas, mantendo apenas o valor máximo destas, reduzindo o tamanho do dado, mas mantendo as informações significantes (Karkare, 2019) (Figura 8).

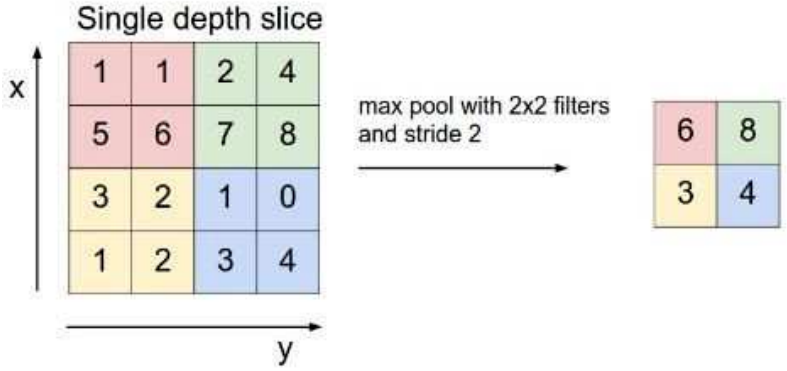


Figura 8 - Exemplo de um max polling (Karkare, 2019)

A estrutura padrão de uma CNN se dá pela alternação de camadas de convolução e *pooling*. Com essas combinações de convolução com *pooling*, é possível extrair diferentes características da imagem, com um menor custo computacional (Karkare, 2019).

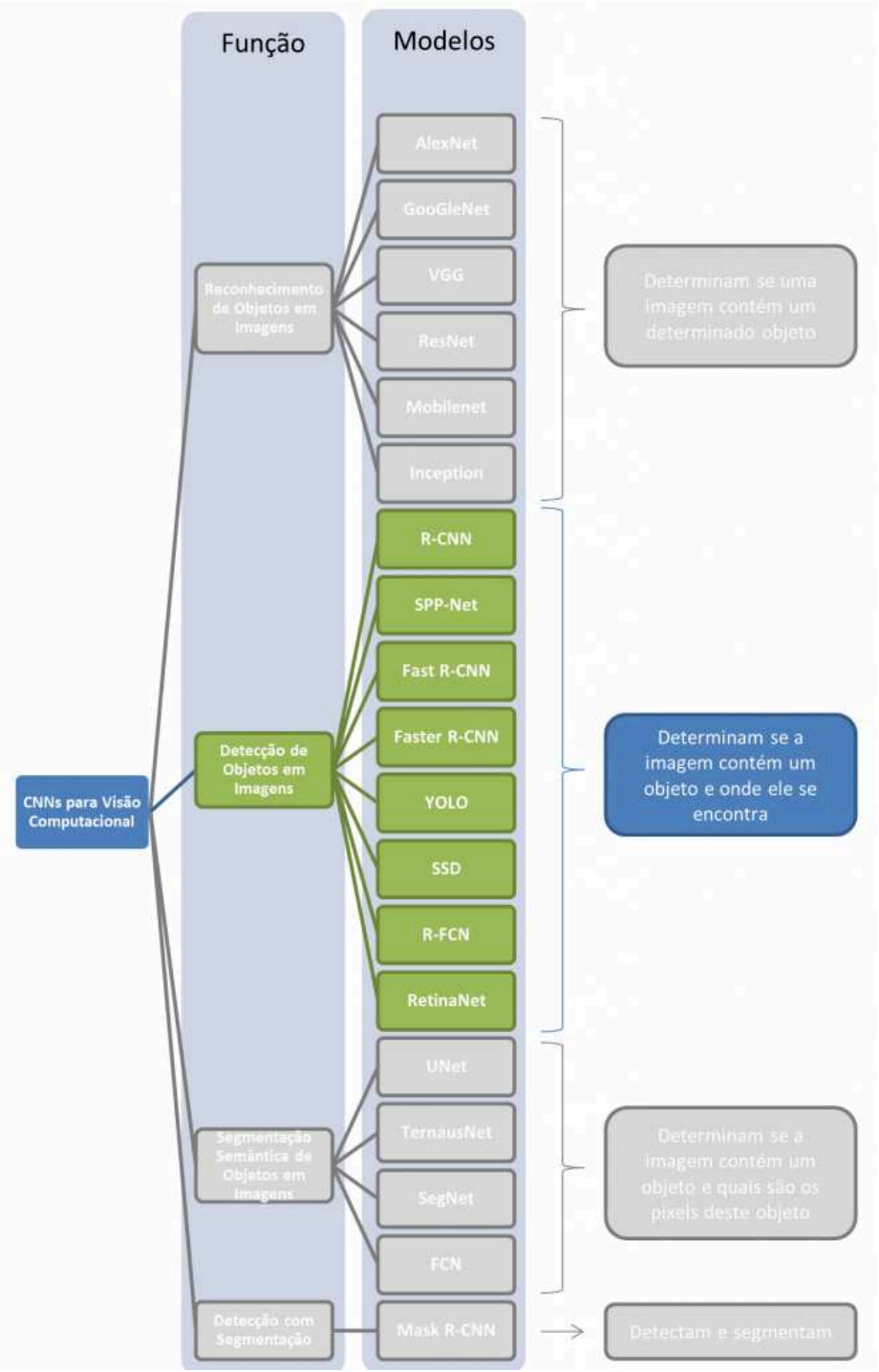


Figura 9 - Redes neurais para visão computacional (von Wangenheim, 2018)

O modelo de redes neurais convolucionais englobam diversas arquiteturas, que variam em função da ativação dos nós, tipos de camadas etc. A Figura 9 mostra um esquema com comparação de diversos modelos utilizados para visão computacional. Como o nome “redes neurais convolucionais” indica, esta classe de redes faz uso da operação matemática convolução. Redes neurais convolucionais

são simplesmente redes neurais que usam a convolução no lugar de multiplicação de matrizes em ao menos uma de suas camadas (Goodfellow et al., 2016).

Métodos de Aprendizado. A forma mais comum do *DL* é o aprendizado supervisionado (Lecun et al., 2015). O qual os seres humanos devem transferir seu conhecimento para o conjunto de dados para que uma rede neural aprenda a correlação entre rótulos e dados (Pathmind, 2019a). No treinamento supervisionado, um sistema de *machine learning* recebe dados previamente categorizados de maneira correta, e é dividido em duas categorias: Classificação e Regressão (Pant, 2019). Em problemas de classificação, a variável alvo é categórica, ou seja, os dados são classificados na classe A, classe B, ou alguma outra classe. Em problemas de regressão, a variável alvo é contínua (numérica) (Pant, 2019) conforme mostrado na Figura 10.

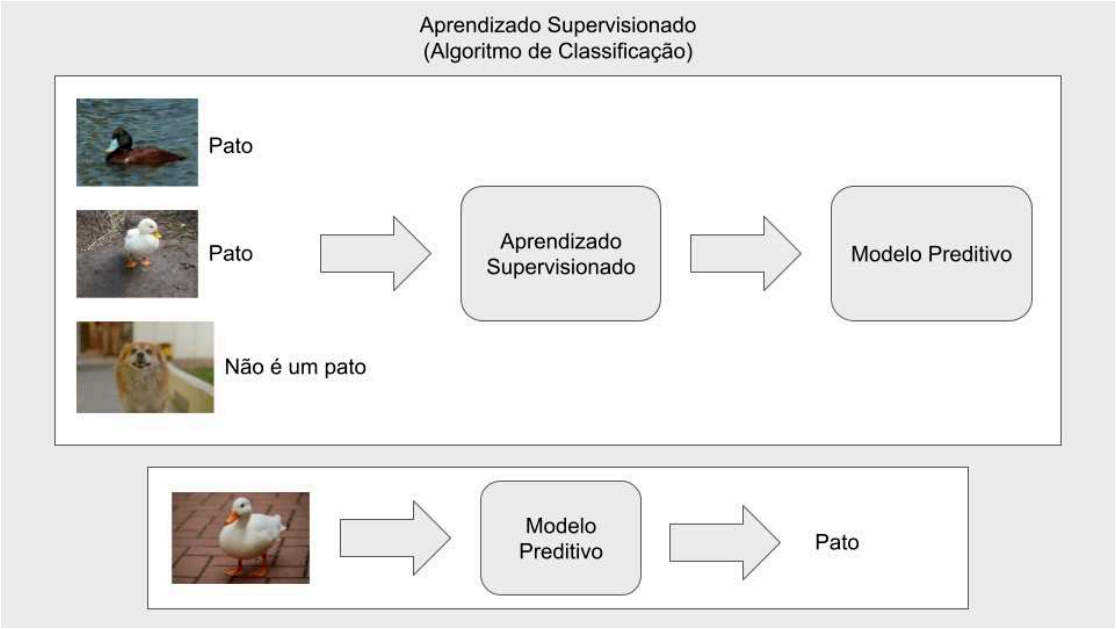


Figura 10 - Exemplo de Aprendizado Supervisionado

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

Durante a criação de uma rede neural, os pesos de suas ligações geralmente recebem valores aleatórios, sendo atualizados ao longo do processo de treinamento por algum algoritmo, de modo a encontrar os valores ideais (Bushaev, 2017). Para se determinar quais valores são melhores que outros, é necessário determinar quão próximo o resultado obtido está do valor esperado, que é geralmente feito por meio de uma função de perda, utilizada devido a sua maior facilidade de minimização quando comparada à maximização da acurácia (Bushaev, 2017).

O problema de treinamento é então equivalente ao problema de minimização da função perda, o que pode ser feito utilizando diversos algoritmos, com gradiente descendente estocástico sendo o mais utilizado (Bushaev, 2017).

Detecção de Objetos. A detecção de objetos é um grande desafio na visão computacional e tem recebido uma grande atenção nos últimos anos, devido à sua ampla gama de recursos e aplicações do mundo real. Especialmente abordagens de aprendizado de máquina, alcançaram avanços notáveis usando técnicas de reconhecimento de padrão, bem como *deep learning* (Jiao et al., 2019). As CNNs são amplamente usadas para detecção de objetos (Zhao et al., 2019). Atualmente, as abordagens de detecção de objetos baseadas em CNN podem ser divididas principalmente em dois tipos: detectores estruturados de dois estágios, como CNN com base na região (*R-CNN*) e um estágio, detectores puramente neurais, como *RetinaNet* e *YOLO* e suas variantes (Wu et al., 2020). A detecção de objetos trata da detecção de instâncias de objetos semânticos de uma determinada classe (como humanos, móveis, animais ou carros) em imagens e vídeos digitais. Diferentes tipos de aprendizagem podem ser aplicados, incluindo abordagens supervisionadas e não supervisionadas *(*Russel et al., 1995*).*

A detecção de objetos inclui a proposta de região, representação de recurso e classificação de região (Hechun e Xiaohong, 2019). As primeiras localizações possíveis do objeto na imagem são sugeridas, indicando também alguma possível área candidata do objeto contido. Em seguida, os recursos são extraídos selecionando os recursos apropriados como vetor de características e classificando o vetor de recursos e, assim, determinando seu tipo (Hechun e Xiaohong, 2019).

Depois de executar as operações de pós-processamento, como a inibição máxima, o retorno da posição da borda, o quadro final do objeto é retornado (Hechun e Xiaohong, 2019) conforme demonstrado na Figura 11.

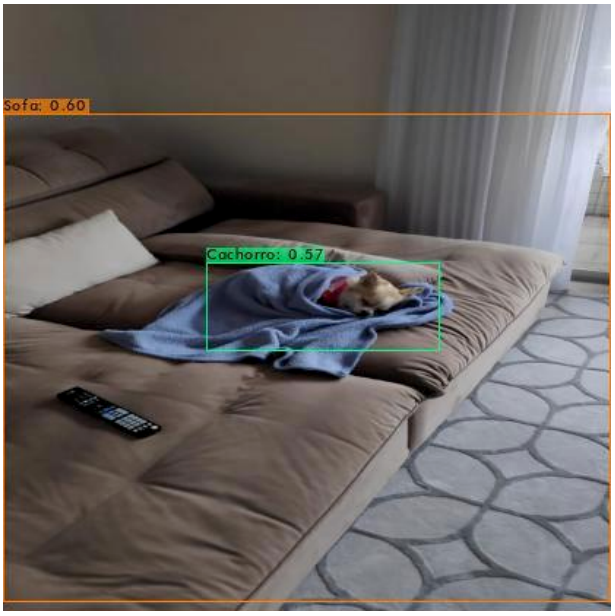


Figura 11 - Exemplo de detecção de objetos

A arquitetura *YOLO* transforma o problema de detecção em um problema de regressão, gerando probabilidades de coordenadas e caixas delimitadoras para cada classe (Costa et al., 2020). Essa característica aumentou consideravelmente a sua velocidade de detecção em comparação com outras redes. A rede divide cada imagem do conjunto de treinamento em grades S × S. Se o centro do objeto de destino estiver em uma grade, a grade será responsável por detectar o alvo. Cada grade fornece as caixas delimitadoras e seu respectivo índice de confiança e probabilidades condicionais da classe (Costa et al., 2020). Portanto, a saída da rede é um conjunto de caixas delimitadoras com sua respectiva confiança. De modo geral, predições com confiança abaixo de 50% são descartadas (Costa et al., 2020) conforme mostrado na Figura 12.

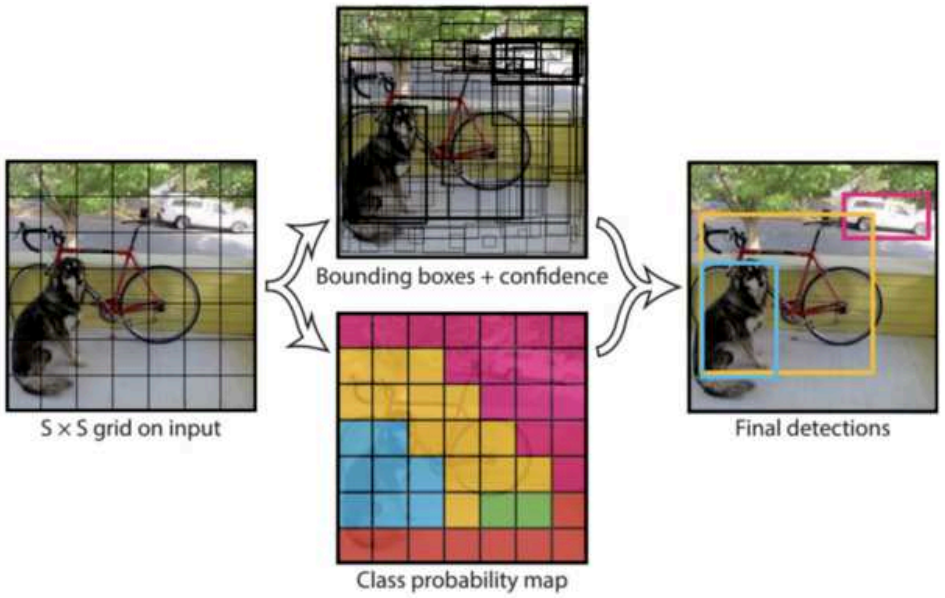


Figura 12 - Exemplos do comportamento *YOLO*

Esses modelos de redes neurais demonstrados na Figura 12 servem tanto para identificar objetos como a região onde os objetos se encontram. Tipicamente a saída dessas redes é um caixa delimitadora (*bounding box*) que registra a região de maior confiança da presença do objeto detectado (von Wangenheim, 2018). Entre os diferentes conjuntos de dados usados pela detecção de objetos, é utilizado diversas métricas para medir o desempenho do modelo de rede neural, entre as mais comuns estão: *mAP*, *IOU*, *recall*, *precision*, *F1 Score* etc. (Padilha et al., 2020).

Tabela 1 - Principais métricas de desempenho para detecção de objetos

Principais métricas para detecção de objetos	**Definição**
Precision	*Precision* é a métrica que informa a capacidade de um modelo identificar apenas objetos relevantes. É a porcentagem de acerto em relação às previsões positivas.
Recall	*Recall* é a métrica que informa a capacidade de um modelo de encontrar todos os casos relevantes (caixas delimitadoras de verdade total). É a porcentagem de previsões corretas positivas entre todas as verdades básicas dadas.
IoU (Intersection Over Union)	Ela é uma medida baseada no Índice *Jaccard* que avalia a sobreposição entre duas caixas delimitadoras, e é usada para determinar se a caixa delimitadora foi prevista corretamente.
F1 Score	F1 Score é a média harmônica entre *precision* e *recall*. O intervalo do F1 Score fica entre 0 e 1. Ele informa o quão preciso é o seu modelo, bem como o quão robusto ele é. Com a *precision* alta mas com baixo *recall*, fornece uma alta taxa de precisão na detecção, mas perde um número significativo de instâncias difíceis de detectar. Quanto maior o *F1 Score*,

	melhor é o desempenho do modelo.
AP	AP é uma métrica popular na medição da precisão de detectores de objetos.
mAP	O mAP é uma métrica usada para medir a precisão de detectores de objetos em todas as classes em um *dataset* específico. O mAP é simplesmente o AP médio de todas as classes

Processo de *Machine Learning.* O modelo de *ML é* um algoritmo, que se torna inteligente após ser treinado com um conjunto de dados. Construir um aplicativo de ML de maneira interativa e centrada no ser humano é um processo iterativo que exige que os alunos executem uma sequência de etapas, conforme apresentado na Tabela 2 (Amazon, 2019; Amershi et al. 2019; Mathewson 2019; Watanabe et al. 2019; Fiebrink e Gillies, 2018).

Tabela 2 - Etapas do aprendizado de máquina (GRESSE VON WANGENHEIM e VON WANGENHEIM, 2021)

Análise de requisitos do modelo ML		Durante esta fase, o objetivo principal do modelo DL e seus recursos de destino são especificados. Isso também inclui a caracterização das entradas e saídas esperadas, especificando o problema e o desempenho esperado.
Preparação de dados	**Coleta e limpeza de dados**	Durante a coleta de dados, conjuntos de dados potencialmente disponíveis são identificados e/ou os dados são coletados. Os dados são limpos, o que envolve a remoção de registros imprecisos ou com ruído do conjunto de dados. Os dados também são padronizados quanto ao formato e tamanho do arquivo (formato e tamanho do arquivo, p.ex. .jpeg e 640x640 pixels)
	Rotulagem de dados	A anotação envolve a atribuição de rótulos e a localização dos bounding boxes na imagem a cada registro para o aprendizado supervisionado.
	Pré-processamento de dados	Para aumentar o conjunto de dados, normalmente são realizados aumentos, como cortar, girar etc. O conjunto de dados é normalmente dividido em um conjunto de treinamento para treinar o modelo, um conjunto de validação para selecionar o melhor candidato de todos os modelos e um conjunto de teste para realizar uma avaliação de desempenho imparcial do modelo. A verificação da qualidade dos dados é realizada a fim de verificar se os objetivos podem ser alcançados com a dada qualidade dos dados disponíveis (dados incorretos ou insuficientes, desequilibrados ou enviesados).
Treinamento e avaliação de modelo	**Transferência de treinamento**	Um modelo é construído ou, mais tipicamente, escolhido a partir de modelos conhecidos que se mostraram eficazes em problemas ou domínios comparáveis. O treinamento de transferência é sobre como aproveitar as representações de recursos de um modelo pré-treinado, que geralmente são treinados em conjuntos de dados massivos que são uma referência padrão (como *COCO (https://cocodataset.org/)*) reutilizando os pesos obtidos em outras tarefas. Incluir os modelos pré-treinados em um novo modelo leva a um menor tempo de treinamento e menor erro de generalização. É particularmente útil quando você tem um pequeno conjunto de dados de treinamento. Nesse caso, você pode, por exemplo, usar os pesos dos modelos pré-treinados para inicializar os pesos do novo modelo. Definir rotinas de treinamento envolve definir *o* número de *épocas, tamanho do lote* e *tamanho de imagem* bem como as métricas de desempenho.
	Avaliação de desempenho	O modelo treinado é avaliado com base nas informações sobre a taxa de perda / erro durante o treinamento, bem como métricas de desempenho específicas dependendo da tarefa específica de DL a ser alcançada, como *IoU (Intersection Over Union), (Average Precision - AP), Precision e Recall, F1 score* etc. (Zou et al. 2019)(Liu et al. 2020)(Robinson, 2018).
Predição		O modelo é testado com novos dados para obter uma aproximação de como o modelo se comporta no mundo real.
Exportação de modelo		O modelo é exportado para permitir sua integração em um sistema de software. A exportação de modelo pode ser no formato específico do modo ou usando formatos abertos como *ONNX* (onnx.ai), *tflite (tensorflow.org/lite), torchscript (pytorch.org), pt (pytorch.org) ou pb (tensorflow.org)* criado para representar diversos tipos de modelos de aprendizado de máquina.

Frameworks e bibliotecas de *DL*. F*rameworks* e bibliotecas são códigos disponibilizados com o propósito de serem utilizados por terceiros para resolver problemas comuns, eliminando a necessidade de retrabalho. Bibliotecas são invocadas por programadores em seu código, enquanto um *framework* já possui todo fluxo de controle, restando ao programador apenas o preenchimento das lacunas restantes. Assim no contexto do *DL* atualmente é utilizado diversos *frameworks* e bibliotecas conforme apresentado na Tabela 3.

Tabela 3 - Frameworks e Linguagens Compatíveis

Framework	**Linguagens Compatíveis**
Tensorflow	C++, Python, Javascript
Torch	C, LuaJIT
Pytorch	C++, Python
Keras	Python
Caffe	C++, Python
The Microsoft Cognitive Toolkit (CNTK)	C++, C#, Python, Java
fastai	Python
Darknet	C++, Python

Cada *framework* possui suas vantagens e desvantagens, tornando-os opções viáveis dependendo de sua aplicação. Contudo, *frameworks* requerem um esforço maior por parte do usuário para entender e fazer uso de suas funções quando comparados a bibliotecas. Para solucionar isso, foram desenvolvidas bibliotecas com base nesses *frameworks*, visando aumentar o nível de abstração e assim facilitar o seu uso e tornar o poder do estado-da-arte em *deep learning* mais acessível e disponível para qualquer pessoa (Fastai, 2019). *Ultralytics/yolov5* (github.com/ultralytics/yolov5) é uma biblioteca que utiliza uma rede de nova geração da série *YOLO* (*You Only Look Once*) usado principalmente para o desenvolvimento de modelos de detecção de objetos. É um produto de integração e inovação contínua baseado em *YOLOv3* e *YOLOv4* e utiliza *pytorch* como *framework,* a rede *YOLOv5* possui oficialmente 5 tipos de modelos: *YOLOv5n,*

YOLOv5s, *YOLOv5m*, *YOLOv5l* e *YOLOv5x* (Zhang et al., 2021). Como demonstrado na figura 13.

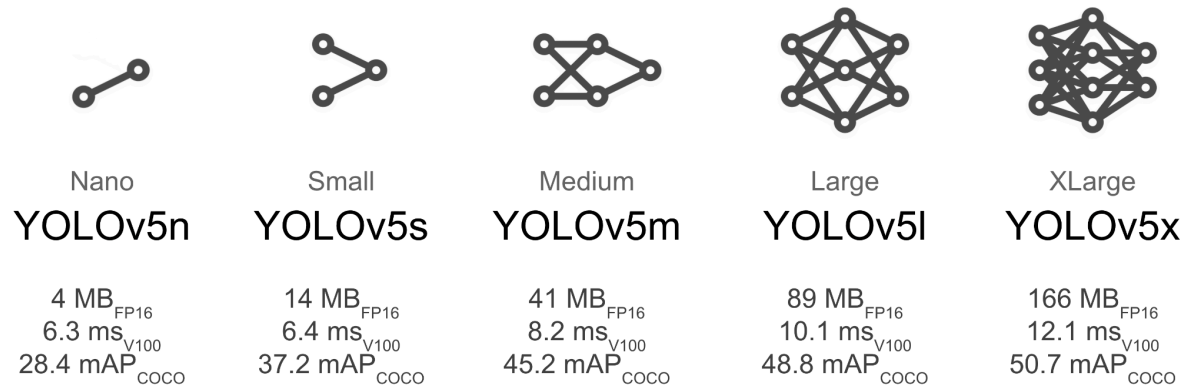


Figura 13. Modelos do YOLOv5 (github.com/ultralytics/yolov5)

2.2. Jupyter Notebooks and Google Colab

O *Jupyter Notebook* (jupyter.org) é uma ferramenta para programação que fornece um ambiente interativo no qual os usuários podem criar documentos que combinam linguagem natural formatada, código executável e multimídia. Isso resulta em uma rica experiência de computação interativa que é difícil de replicar com outras ferramentas (Johnson, 2020). Para o usuário, um bloco de código do *Jupyter Notebooks* aparece como uma sequência de células, cada uma das quais pode conter uma gama de entradas e saídas: *Markdown*, imagens, vídeo, *LATEX* e, o mais importante, fragmentos de código que podem ser executadas pelo usuário individualmente (Johnson, 2020). O *Jupyter Notebook* (jupyter.org) é um aplicativo gratuito de código aberto que permite a combinação de diversas linguagens de programação, como *Python*, *R*, etc. Ele é normalmente usado como ambiente de programação baseada em texto para desenvolver modelos de *ML* usando, por exemplo, *Python*. É um ambiente de computação interativo, acessível através de qualquer navegador da web, que permite aos usuários usar, modificar ou criar documentos da web multimídia interativos. Os documentos do *Jupyter Notebook* fornecem um registro completo e executável de computação que pode ser compartilhado com outras pessoas e fornece um registro completo e executável do processo. Os arquivos de notebook criados são um formato *JSON* simples e

documentado, com a extensão (*.ipynb)*, que pode ser facilmente acessado e manipulado por outras ferramentas de software.

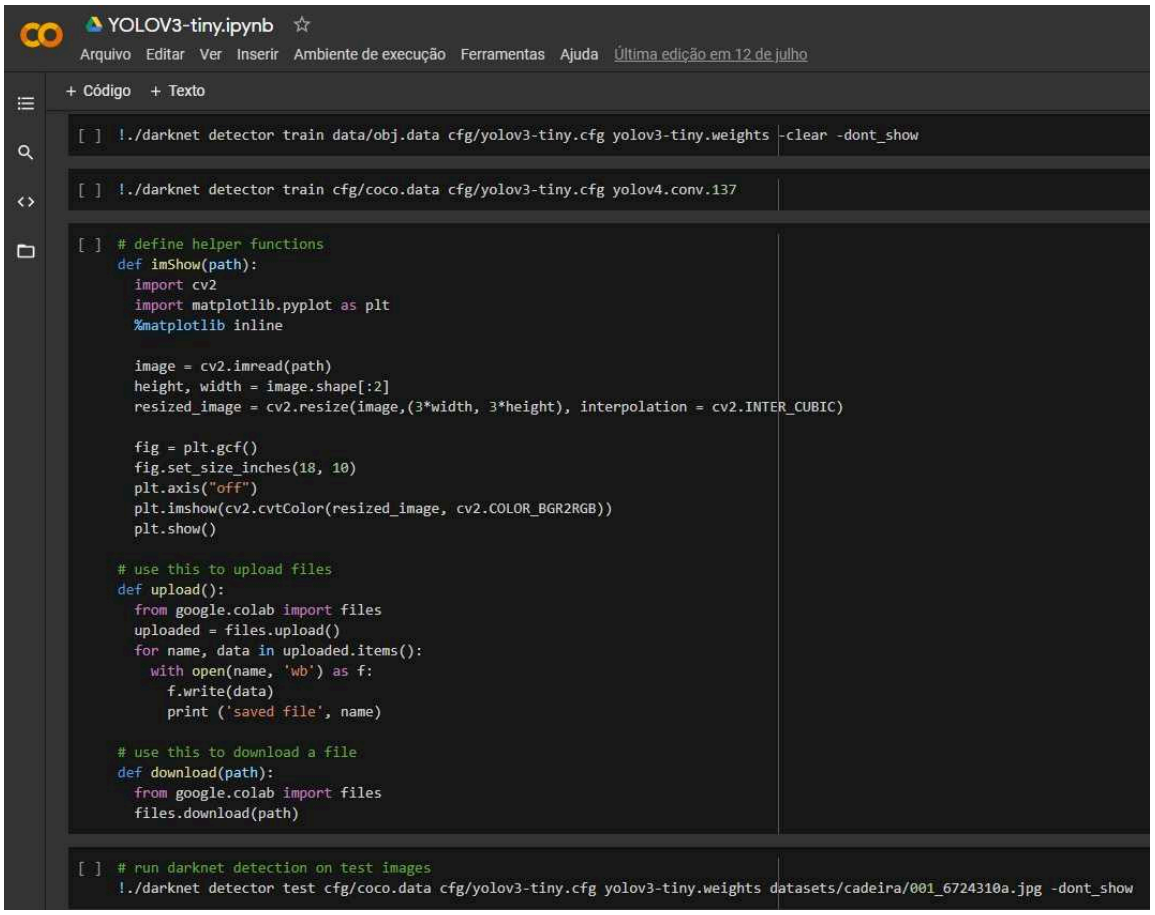


Figura 14 - Exemplo de uma interface textual de um *Jupyter Notebook* no *Google Colab*

O *Jupyter Notebook* é um excelente ambiente de aprendizagem para os alunos (Barba et al., 2019), oferecendo uma ferramenta valiosa para o ensino em direção à compreensão, movimentando os alunos, por exemplo, da visualização passiva do conteúdo do curso para explorar, analisar, sintetizar e criar conteúdo de maneiras ativas. Para começar, os alunos podem consumir o conteúdo do *notebook* lendo e executando no estágio de "use" do ciclo use-modifique-crie (Lytle et al., 2019) e, em seguida, passar para editar ou completar no estágio de "modificação", além de criar suas próprias soluções no estágio de “criação”. Também proporcionará benefícios que impulsionam o engajamento, o interesse e a exploração de conceitos (Barba et al., 2019) e aumentam o desempenho (Freeman et al., 2014). Além disso, os *Notebooks* do *Jupyter* são fáceis de usar, dispensando instalação e podem ser adotados em diversos tipos de cursos (online ou presencial).

O *notebook Jupyter* possui dois componentes. Os usuários inserem código de programação ou texto em células retangulares em uma página da web de front-end. O navegador então passa esse código para um *back-end 'kernel'*, que executa o código e retorna os resultados. É importante ressaltar que os kernels não precisam residir no computador do usuário, pois os *notebooks* também podem ser executados na nuvem. Por exemplo, o *Colaboratory* da *Google* (https://colab.research.google.com). Ele permite que os usuários colaborem e executem códigos inteiramente na nuvem, explorando os recursos de nuvem do *Google*, como unidades de processamento gráfico, e salvem facilmente seus artefatos no *Google Drive*. Isso permite treinar modelos de ML mesmo sem acesso a uma máquina potente ou acesso à Internet de alta velocidade. O *Google Colab* oferece suporte a instâncias de *GPU* e *TPU*, o que o torna uma ferramenta perfeita para aprendizado profundo. Como um *notebook Colab* pode ser acessado remotamente de qualquer máquina por meio de um navegador que não requer configuração para uso, ele também é adequado para fins educacionais. O Colab também permite que os usuários usem e compartilhem *notebooks Jupyter* com outras pessoas sem precisar baixar, instalar ou executar nada, pois os *notebooks Colab* são armazenados no *Google Drive* ou podem ser carregados do *GitHub*. Além disso, também permite fazer upload de artefatos, como imagens diretamente de um *Google Drive*. Isso o torna prático para uso também em contexto educacional, no qual, por exemplo, um professor pode configurar um *notebook* e dar acesso facilmente aos alunos (Kluyver et al., 2016).

Além da forma comum de programação baseada em texto, a extensão *IPyWidgets* (https://ipywidgets.readthedocs.io/en/) são *widgets HTML* interativos para *Jupyter Notebooks*, que permitem uma interface gráfica por meio de menus de seleção, barras deslizantes, botões de rádio/alternância ou caixas de texto. Esses *widgets* gráficos podem ser incorporados aos notebooks Jupyter e os usuários podem combinar a análise de script usual com a ativação de tais *widgets* quando desejado. Eles podem ser usados, por exemplo, para variar os valores dos parâmetros de entrada e explorar um conjunto de dados ou resultados computacionais. Esses elementos podem ocultar o código baseado em texto e

permitir a criação de um bloco de notas com uma interface visual. Atualmente, *IPyWidgets* são usados principalmente para análise e visualização de dados, permitindo transformar *Jupyter Notebooks* em painéis interativos para explorar e visualizar dados (Perkel, 2018; Piazentin Ono et al., 2021).

2.3. Ensino de Machine Learning no Ensino Superior

O estudo da Inteligência Artificial prepara o aluno para determinar quando uma abordagem de *IA* é apropriada para um determinado problema, identificar a representação apropriada e o mecanismo de raciocínio, e implementá-lo e avaliá-lo (ACM, 2013). De acordo com a ACM os conceitos que precisam ser cobertos em *Machine Learning* no curso de computação no ensino superior são divididos em duas partes, básico e avançado (ACM, 2013). A parte básica inclui:

- Definir uma ampla variedade de tarefas de *ML*, incluindo classificação e detecção de objetos.
- O problema do *overfitting.*
- Medir a precisão do modelo gerado.

Já a parte avançada do ensino de *Machine Learning* no curso de computação inclui os seguintes tópicos:

- Definição e exemplos de uma ampla variedade de tarefas de *Machine Learning*
- Aprendizagem baseada em estatísticas gerais, estimativa de parâmetros
- Programação lógica indutiva
- Aprendizagem supervisionada
    - árvores de decisão
    - Redes neurais
    - Máquinas de vetor de suporte
- Algoritmos do vizinho mais próximo
- Aprendizagem não supervisionada e agrupamento
    - Mapas auto-organizáveis
- Aprendizagem semi supervisionada

- Modelos gráficos de aprendizagem
- Avaliação de desempenho
- Teoria de aprendizagem
- O problema do *overfitting*, a maldição da dimensionalidade
- Aprendizagem por reforço
    - Negociação entre exploração e exploração
    - Processos de decisão de Markov
    - Valor e iteração de política
- Aplicação de algoritmos de *Machine Learning* para mineração de dados

Ao final dos cursos de *Machine Learning* espera-se que os estudantes de Ciências da Computação tenham adquirido as seguintes competências (ACM, 2013):

- Explicar as diferenças entre os três estilos principais de aprendizagem (supervisionado, reforçado e não supervisionado)
- Consiga implementar algoritmos simples para aprendizagem supervisionada, aprendizagem por reforço e aprendizagem não supervisionada
- Determinar qual dos três estilos de aprendizagem é apropriado para um domínio de um problema particular
- Avaliar o desempenho de um sistema de aprendizagem simples em um *conjunto de dados*
- Comparar cada uma das seguintes técnicas, fornecendo exemplos de quando cada estratégia é superior: árvores de decisão, redes neurais e redes de crenças.
- Avaliar o desempenho de um sistema de aprendizagem simples.
- Explicar o problema de *overfitting*, juntamente com técnicas para detectar e gerenciar o problema.

O desenvolvimento de aplicativos de *ML* não é trivial e o processo de desenvolvimento difere de um software tradicional, pois envolve adquirir um conjunto rotulado de exemplos, selecionar um algoritmo de aprendizado apropriado

e seus parâmetros, treinar um modelo, avaliar as previsões deste modelo em relação ao conjunto de testes e, finalmente, sua implantação em uso (Ramos et al., 2020). Normalmente, os modelos de *ML* são desenvolvidos usando linguagens de programação baseadas em texto que requerem codificação (Tamilselvam et al. 2019). Para democratizar o aprendizado de *ML*, é desejável reduzir significativamente o esforço cognitivo, permitindo ao estudante se concentrar na lógica para resolver o problema. Observa-se uma curva acentuada de aprendizado envolvida no entendimento dos fundamentos de *ML* e das nuances sintáticas de várias bibliotecas utilizadas (Tamilselvam et al. 2019). Essas dificuldades são ainda mais salientes em algoritmos e bibliotecas para *Deep Learning (DL)*. Assim, necessita-se de ferramentas intuitivas para criar, de forma mais fácil, modelos de *ML* em geral e *DL* (Tamilselvam et al. 2019).

2.4. Ambientes visuais para desenvolvimento de modelos ML

Recentemente a programação visual está cada vez mais presente na programação com diversas ferramentas como *Blockly(https://blockly.games/), Scratch(https://scratch.mit.edu/), Snap!(https://snapml.readthedocs.io/), App Inventor(https://appinventor.mit.edu/)*, os quais aparecem também em estudos para uma possível implementação nos primeiros passos do aluno no ensino superior (Weintrop et al., 2019). Tem o intuito acelerar a absorção do conhecimento pelo aluno, não tendo o objetivo de fazer por fazer, mas ensinar o aluno os conhecimentos básicos sem os problemas de sintaxes existentes (Weintrop et al., 2019).

O desenvolvimento de aplicativos de ML não é trivial e o seu processo de desenvolvimento difere de um software tradicional, pois envolve adquirir um conjunto rotulado de exemplos, selecionar um algoritmo de aprendizado apropriado e seus parâmetros, treinar um modelo, avaliar as previsões deste modelo em relação ao conjunto de testes e, finalmente, sua implantação em produção (Ramos et al., 2020).

Para o desenvolvimento de aplicações de IA com ferramentas visuais é possível identificar diferentes tipos de ferramentas de suporte para o desenvolvimento de modelos de ML (Gresse von Wangenheim et al., 2021) Ambientes de programação baseados em blocos, como por exemplo o *eCraft2Learn* (Kahn e Winters, 2018), em que num ambiente sem nada os blocos devem ser encaixados como um quebra-cabeça para montar a solução desejada como demonstrado na Figura 15.

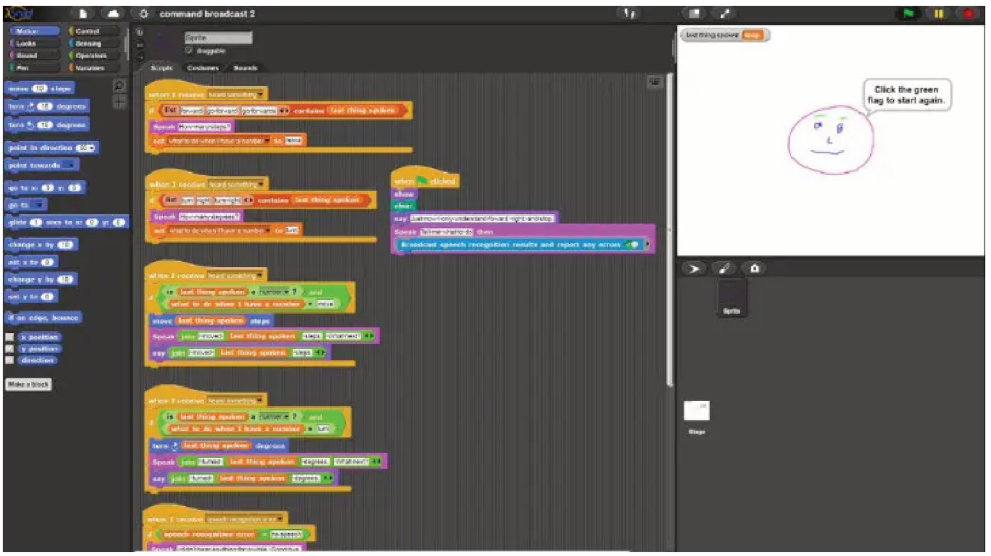


Figura 15 - Programação em blocos (ecraft2learn.com)

Outra alternativa são ambientes de programação baseados em fluxo de dados, que são linguagens visuais em que blocos (também chamados de nós) são conectados por arcos (ou fios). Portanto, um programa em uma linguagem de programação baseada em fluxo é um gráfico direcionado por meio do qual os dados fluem entre os blocos e cada bloco fornece uma função que pode transformar os dados recebidos (Johnston et al., 2004) (Hils, 1992).

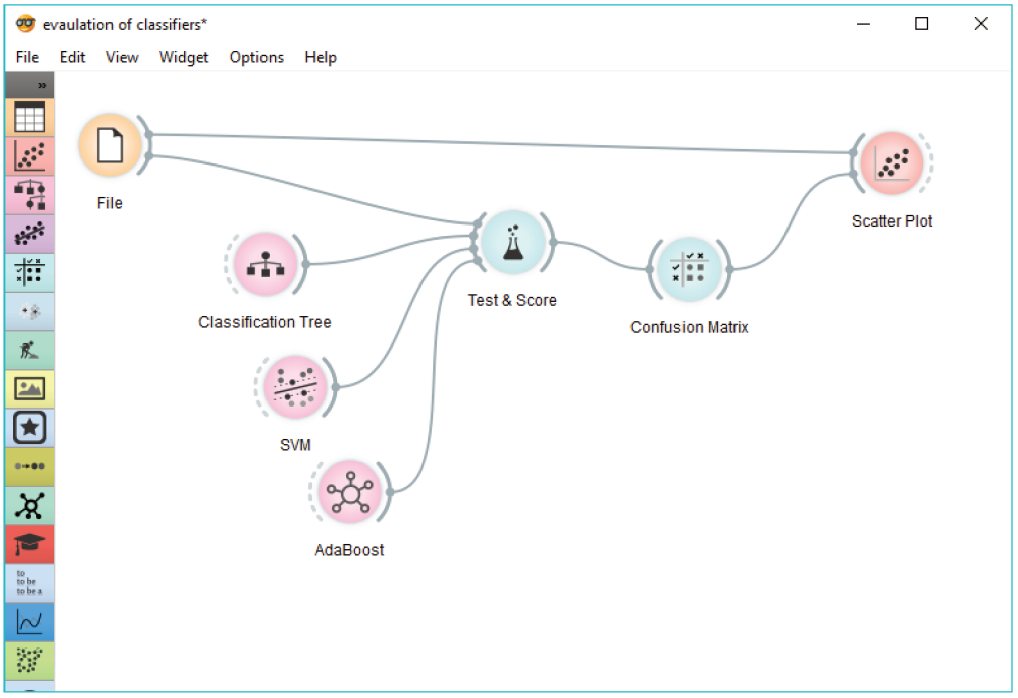


Figura 16 - Programação baseado em fluxo de dados (https://orangedatamining.com/)

Estes ambientes baseados em fluxo por sua vez podem ser divididos em dois tipos: Ambientes de programação baseados em fluxos de trabalho, que normalmente possuem o fluxo de trabalho em si bem definido, e apresentam dentro dos blocos as etapas necessárias e também os ambientes de programação baseados em fluxos de dados, em que é possível montar da forma que se desejar todo o processo idealizado, não ficando necessariamente preso as etapas definidas pelo criador da ferramenta como demonstrado na Figura 17.

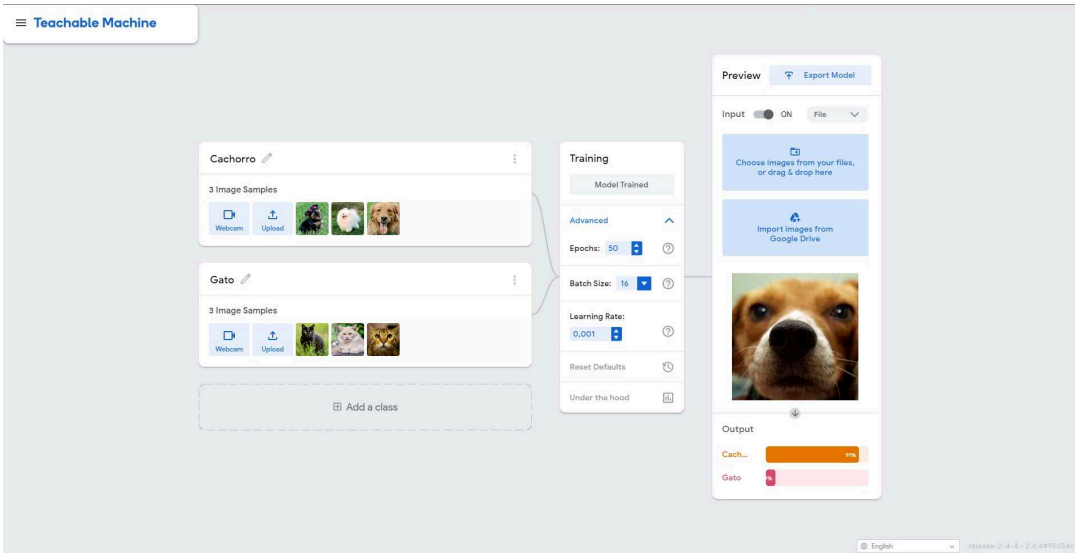


Figura 17- Programação em Fluxos (https://teachablemachine.withgoogle.com/)

3. ESTADO DA ARTE

Para levantar o estado da arte, foi conduzido um mapeamento sistemático seguindo os procedimentos propostos por Petersen et al. (2008).

3.1. Definição do protocolo de revisão

O objetivo da realização deste mapeamento sistemático é responder a seguinte pergunta da pesquisa**:** Quais ferramentas visuais existem para ensinar conceitos de *Machine Learning* no contexto do ensino superior? Essa pergunta de pesquisa é refinada nas seguintes questões de análise:

AQ1: Quais ferramentas visuais existem para ensinar *ML* no ensino superior ?

AQ2: Quais são as suas características em relação à plataforma de *ML*?

AQ3: Quais são as características delas em relação à plataforma de deployment?

AQ4: Como as ferramentas de *ML* foram desenvolvidas e avaliadas?

Critérios de inclusão/exclusão: Foram consideradas ferramentas para o desenvolvimento e deployment de modelos *ML* que foram aplicados pelo menos uma vez no ensino superior, publicadas entre 2010 e 2020. Ferramentas que não abordam conceitos de *ML* e/ou não adotam um paradigma visual não foram consideradas. Foram excluídas também ferramentas focadas ao ambiente profissional. Foram consideradas apenas ferramentas para as quais foram encontradas as informações suficientes.

Fontes dos dados: Foram pesquisados os principais bancos de dados e bibliotecas digitais da área de computação, incluindo *ACM Digital Library*, *IEEE Xplore Digital Library*, *Scopus* e *Google Scholar* com acesso via Portal da Capes. Foi realizada também uma busca no Google, a fim reduzir o risco de omissão de ferramentas não publicadas em bibliotecas científicas.

Definição da Search String: 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. Usou-se também sinônimos

apresentados na Tabela 4 para minimizar o risco de omissão de trabalhos relevantes.

Tabela 4 - Termo de Busca e seus Sinônimos

Termo de Busca	Sinônimo(s)
Visual Programming	block-based programming, block-based
Machine Learning	deep learning, neural network, neural networks, AI, Artificial Intelligence
higher education	university, undergraduate, college , introductory, novice learners

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

(“visual programming” OR “block-based programming” OR “block-based” or “approachable interface”) AND (“machine learning” OR “deep learning” OR “neural network” OR “ai” OR “artificial intelligence”) AND (“higher education” OR “undergraduate” OR “college” OR “introductory” OR “novice learners”)

Dessas palavras-chave, a search string foi adaptada para cada fonte de dados apresentada na Tabela 5.

Tabela 5 - Search string para cada base de dados

Base de Dados	String de Busca
ACM Digital Library	[[All: "visual programming"] OR [All: "block-based programming"] OR [All: "block-based"] OR [All: "drag-and-drop"] OR [All: "toolkit"] OR [All: "gui tool"]] AND [[Abstract: "machine learning"] OR [Abstract: "deep learning"] OR [Abstract: "neural network"] OR [Abstract: "ai"] OR [Abstract: "artificial intelligence"]] AND [[All: "higher education"] OR [All: "undergraduate"] OR [All: "college"] OR [All: "introductory"] OR [All: "students"]] AND [Publication Date: (01/01/2010 TO 12/31/2020)]
IEEE Xplore Digital Library	(("All Metadata": "visual programming" OR "All Metadata": "block-based programming" OR "All Metadata": "block-based" OR "All Metadata": "drag-and-drop" OR "All Metadata": "approachable interface" OR "All Metadata": "gui tool" OR "All Metadata": "toolkit") AND ("Abstract": "machine learning" OR "Abstract": "deep learning" OR "Abstract": "neural network" OR "Abstract": "ai" OR "Abstract": "artificial intelligence") AND ("All Metadata": "higher education" OR "All Metadata": "undergraduate" OR "All Metadata": "college" OR "All Metadata": "introductory" OR "All Metadata": "user studies" OR "All Metadata": "students"))
Scopus	(TITLE-ABS-KEY ("visual programming" OR "block-based programming" OR "block-based" OR "approachable interface") AND TITLE-ABS-KEY ("machine learning" OR "deep learning" OR "neural network" OR "air" OR "artificial intelligence") AND ALL ("higher education" OR "undergraduate" OR "college" OR "introductory" OR "novice learners"))
Google Scholar	"machine learning" "visual programming" "undergraduate"
Google	"machine learning" "visual programming" "undergraduate"

3.2. Execução da busca e extração dos dados

A pesquisa foi realizada em Novembro de 2020. A busca inicial recuperou um total de 885 artigos nas bases científicas e 178.000 artigos no Google (Tabela 5). Devido ao grande número de resultados de algumas pesquisas, foi restringido a análise às 300 mais relevantes. Publicações irrelevantes e duplicadas retornadas por múltiplas pesquisas foram removidas. Na primeira fase de análise, títulos e resumos foram analisados. Na segunda etapa de seleção, ao analisar o texto completo, aplicam-se os critérios de inclusão e exclusão para identificar os relevantes. Como resultado deste estágio foram identificados 19 artigos potencialmente relevantes. Na terceira etapa foi verificado se os artigos seguiam foi analisado o texto completo aplicando os critérios de inclusão e exclusão para identificar os documentos relevantes. Foram excluídos desta pesquisa quaisquer ferramentas voltadas a outros níveis educacionais (ensino infantil, fundamental e superior) ou que não especificasse o ensino superior.

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

Fonte	Número de resultados da busca	Número de resultados analisados	Número de documentos potencialmente relevantes	Número de documentos relevantes
ACM Digital Library	331	300	9	1
IEEE Xplore Digital Library	34	34	6	1
Scopus	95	95	4	0
Google Scholar	425	300	3	0
Google	178.000	300	2	1
Total (sem duplicados)	178885	960	19	3

Outros ambientes que possuem o potencial para ser apropriados para o ensino superior como, p.ex., Orange (https://orangedatamining.com/), Microsoft Lobe (https://www.lobe.ai/) e Amazon Rekognition (https://aws.amazon.com/pt/rekognition), não foram encontradas evidências de que essas ferramentas foram projetadas e/ou aplicadas neste nível educacional. Consequentemente não foram incluídos nesta revisão. Outros casos foram retirados por não utilizarem uma ferramenta visual a qual, possibilita acompanhar o treinamento do modelo testando as possibilidades como alterar *batch size*, *learning rate* e número de épocas entre outros, e com isso verificar o que acontece ao treinar o modelo com outros valores para os parâmetros e poder acompanhar, testar e validar se o modelo criado está genérico e com uma precisão de acertos aceitavel.

Como resultado final foram identificados 3 ambientes visuais voltados ao desenvolvimento de ML projetado ou aplicado no âmbito do ensino superior conforme apresentado na Tabela 7.

Tabela 7 - Artigos relevantes

Resultado da busca	
Título do artigo	**Referência**
Milo: A visual programming environment for Data Science Education	A. Rao, A. Bihani, and M. Nair. 2018. Milo: A visual programming environment for Data Science Education. In Proceedings of the Symposium on Visual Languages and Human-Centric Computing, Lisbon, Portugal, 211-215.
Google Teachable Machine	M. Carney, B. Webster, I. Alvarado, K. Phillips, N. Howell, J. Griffith, J. Jongejan, A. Pitaru, and A. Chen. 2020. Teachable Machine: Approachable

	Web-Based Tool for Exploring Machine Learning Classification. In Proc. of Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems. ACM, New York, NY, USA, 1–8.
Deep Learning Programming by All	K. M. Kahn and N. Winters. 2018. AI Programming by Children. In Proceedings of the Conference on Constructionism, Vilnius, Lithuania.

3.3. Análise dos trabalhos relacionados

Nesta seção analisamos as ferramentas relevantes para cada pergunta da análise, com objetivo de identificar e metrificar os pontos positivos e negativos de cada uma no âmbito do ensino superior.

3.3.1. Quais ferramentas visuais existem para ensinar *ML* no ensino superior?

Foram identificadas 3 ferramentas visuais de programação desenvolvidas ou sendo usadas para ensinar o processo de criação de modelos personalizados de *ML* no ensino superior (Tabela 8). Comparando-os com as propostas de ferramentas visuais para os estágios como: Educação básica, fundamental e médio, se observa uma menor oferta na qual tipicamente adotam-se ferramentas textuais para o ensino (Weintrop, 2019). Porém, há relatos do uso de ferramentas visuais para o primeiro contato com programação (Weintrop, 2019).

Tabela 8 - Ferramentas visuais para ensinar ML no ensino superior

Nome	Descrição breve	Tarefa(s) de *ML* suportada(s)	Escopo	
			Plataforma de *ML*	Plataforma para implantação
Milo	Um ambiente de programação visual baseado na web com blocos para Educação em Ciência de Dados.	Clustering	X	-
Google Teachable Machine	Um ambiente de programação visual baseado na web para Educação	Classificação de imagens, sons e poses	X	-
eCraft2learn	Blocos adicionais para a linguagem de programação visual Snap! que fornece interface fácil de usar para os serviços de nuvem de *IA* e funcionalidade de aprendizado profundo.	Reconhecimento de imagem e fala, síntese de fala, detecção de objetos e segmentação	X	Snap!

As ferramentas analisadas oferecem suporte parcial e/ou total para o desenvolvimento de modelos de *ML* bem como sua implantação como parte de artefatos de software, como por exemplo em jogos ou aplicativos móveis. A maioria das ferramentas oferece a implantação integrada a ambientes de programação baseados em blocos comumente utilizados no ensino superior como o Snap!. Os ambientes visuais para programação em sua maioria são tipicamente voltados para a educação básica, destaca-se o SNAP!, Scratch, Blockly e App Inventor por serem também utilizados nas fases iniciais para os cursos de computação (Weintrop, 2019).

3.3.2. Quais são as suas características em relação à plataforma de *ML*?

Foram identificados três tipos de ferramentas de suporte para o desenvolvimento de modelos de *ML*. Duas ferramentas (eCraft2Learn, Milo) fornecem suporte baseado em blocos ao estender o respectivo ambiente de programação, fornecendo blocos de *ML* específicos para preparação, treinamento e avaliação de dados. O Google Teachable Machine adota uma abordagem baseada em *workflow* para apoiar o desenvolvimento do *ML*, guiando o usuário passo a passo por meio de um navegador web.

Todas as ferramentas são limitadas quanto à tarefa de *ML* que suportam, focando principalmente no reconhecimento de imagens, sendo as tarefas em que os aplicativos de *ML* atuais estão tendo muito sucesso. Apenas o eCraft2learn recentemente adicionou blocos de suporte à detecção e segmentação de objetos. Outras tarefas cobertas incluem síntese de voz (eCraft2Learn), clustering (Milo) e reconhecimento de posição do corpo e de som (Google Teachable Machine).

As ferramentas suportam amplamente todas as etapas básicas do processo de desenvolvimento de *ML*: primeiro, elas incentivam a coleta de pequenas quantidades de dados e sua rotulagem, organizando-os em categorias criadas pelo usuário. Em seguida, esses dados são usados para treinar um modelo de *ML* usando o aprendizado por transferência, o que permite construir modelos precisos de forma que economize tempo, usando back-ends de *ML* como por exemplo o

TensorFlow. Uma vez que o modelo é treinado, seu desempenho pode ser avaliado. Isso é feito principalmente permitindo que o usuário teste o modelo com novos dados para os quais o modelo fornece como saída o rótulo previsto. Milo e Google Teachable Machine oferecem suporte à exportação do modelo criado para sua implantação em Python (Milo e Google Teachable Machine) e javascript (Google Teachable Machine). Nenhuma das ferramentas oferece suporte à análise de requisitos. A engenharia de recursos também não é oferecida.

As ferramentas suportam uma variedade de tipos de dados, com imagens como o tipo mais frequente. Todas as ferramentas esperam que os usuários coletem seus próprios dados, fazendo com que os alunos se envolvam de forma criativa com os dados incorporando conjuntos com os quais os alunos podem se relacionar e compreender facilmente. A coleta de dados é habilitada via webcam, microfone, arquivos diretamente do computador, etc.

Tabela 9 - Características relativas aos dados disponibilizados pelo usuário

Nome	Tipos de dados	Opções de entrada	Disponibilidade de *datasets* prontos para o uso
eCraft2learn	Imagem, postura e som	Upload de arquivos, webcam e microfone	--
Milo	Números e texto	Upload de arquivo	Poucos conjuntos de dados populares usados em cursos introdutórios de ML, como o conjunto de dados Iris
Google Teachable Machine	Imagem, som	Webcam, microfone, upload de arquivos (Google Drive ou local)	--

Como *back-end* de *ML*, a maioria das ferramentas usa estruturas ou provedores de *ML* comuns, como Tensorflow, ou suas próprias implementações proprietárias. O predomínio do uso do Tensorflow.js pode ser explicado por sua facilidade de execução, sem a necessidade de instalação do lado do cliente ou de infraestrutura dedicada para a ferramenta. Para acelerar o treinamento, algumas ferramentas adotam abordagens de aprendizagem por transferência usando MobileNet ou SqueezeNet como modelos de *deep learning* pré-treinados para reconhecimento de imagem, etc.

Tabela 10 - Características do modelo de ML e aprendizagem

Nome	Back-end/Algoritmos de *ML*	Tipos de aprendizagem	Parâmetros de treinamento

eCraft2learn	Modelos de nuvem pré-treinados, suporte a navegador integrado, IBM Watson	Supervisionado	Iterações de treinamento, taxa de aprendizagem, divisão de validação, embaralhamento de dados
Milo	Tensorflow.js	Supervisionado e não supervisionado	Taxa de aprendizagem, função de perda, métricas de treinamento, iterações
Google Teachable Machine	Tensorflow.js	Supervisionado	Taxa de aprendizagem, métricas de treinamento, iterações

3.3.3. Quais são as características delas em relação à plataforma de *implantação*?

Enquanto várias ferramentas fornecem suporte para a implantação como parte de um jogo ou aplicativo móvel, integrado ou como uma extensão de um ambiente de programação visual, a solução Milo suporta apenas a exportação do modelo de *ML* criado.

Dependendo das tarefas específicas que a ferramenta suporta, blocos de programação de *ML* são fornecidos para incorporar o *ML* criado ao projeto. Dependendo da variedade de tarefas suportadas pela ferramenta, isso pode variar de poucos blocos (como 3 blocos de reconhecimento de imagem) a conjuntos maiores para diversos fins. Em geral, esses novos blocos são projetados em conformidade com o design visual da respectiva linguagem de programação baseada em blocos.

Tabela 11 - Características da Plataforma de Implantação

Nome	**Apenas exportar**	**Suporte ao Implantação em ambientes visuais**	
		Plataforma	**Blocos adicionados para *ML***
eCraft2learn	--	Snap!	Blocos diversos para várias tarefas de *ML*
Milo	Python code	--	--
Google Teachable Machine	Javascript, Python e Kotlin	--	--

Dessa forma, as ferramentas permitem que os alunos aprendam os conceitos de *ML* ao mesmo tempo que os capacita a criar artefatos significativos com impacto direto em suas vidas e comunidades. Isso pode motivá-los a criar aplicativos inovadores e que atendam aos seus interesses.

3.3.4. Como as ferramentas de *ML* foram desenvolvidas e avaliadas?

A maioria das publicações encontradas carece de uma descrição da metodologia de pesquisa adotada para desenvolver as ferramentas de *ML*. O código-fonte de algumas ferramentas está disponível sob licença de código aberto permitindo sua evolução e adaptação.

Tabela 12 - Informações sobre o desenvolvimento da ferramenta

Nome	Metodologia científica	Disponibilidade do código	Licença do código
eCraft2learn	--	https://github.com/ecraft2learn/ai/blob/master/ecraft2learn.js	BSD
Milo	Comparação entre ambiente visual, tabular e textual.	https://miloide.github.io	Apache 2.0 License
Google Teachable Machine	--	https://github.com/googlecreativelab/teachablemachine-community	Apache 2.0 License

No entanto, outros estudos visando a avaliação das ferramentas são relatados. Os fatores avaliados variam desde a eficácia das ferramentas na aprendizagem, usabilidade, utilidade e eficiência dos alunos até a identificação de seus pontos fortes e fracos. As avaliações foram conduzidas como estudos de caso. Os tamanhos das amostras são em sua maioria pequenos. Os resultados desses estudos indicam que as ferramentas ajudam a alavancar o conhecimento de domínio dos alunos para coletar dados, construir modelos, testar e avaliar modelos, permitindo-lhes realizar iterações rápidas para testar hipóteses sobre o desempenho do modelo e reformular seus modelos.

3.4. Discussão

Analisando os resultados que abordavam o ensino superior foram identificadas somente 3 ferramentas utilizadas para ensinar *ML* neste contexto acadêmico. Provavelmente por que no ensino superior já é mais frequente a utilização de linguagens de programação textual, já que comparado com outros contextos há uma quantidade superior de exemplos de ambientes visuais e aplicações para a educação básica (Gresse von Wangenheim et al., 2021). Assume-se que no ensino superior estes ambientes visuais são utilizados mais em

momentos iniciais para ensinar a parte conceitual e no decorrer da progressão da aprendizagem dos estudantes são substituídos por ambientes textuais (Carney et al., 2020).

As ferramentas possibilitam a criação de diferentes modelos personalizados de *ML* focando principalmente em algoritmos de aprendizagem supervisionada, como somente o Milo oferecendo também suporte para a aprendizagem não supervisionada. Oferecem suporte principalmente para classificação e *clustering* de diversos tipos de dados, incluindo imagens, som, texto etc.

Os ambientes em blocos se mostraram mais eficientes para o desenvolvimento do processo de *ML* em geral, em relação aos ambientes em forma de código e tabular (Tamilselvam et al., 2019). Com o resultado os estudantes apresentaram uma maior capacidade para o desenvolvimento de modelos mais sofisticados o suficiente para soluções mais completas (Tamilselvam et al., 2019)

Todas as ferramentas estão disponíveis online, pelo navegador(e Craft 2 learn, Google Teachable Machine), como aplicação *desktop* ou no navegador pela instalação do projeto no caso do Milo. Isto facilita o manuseio das ferramentas visuais, entretanto a conexão contínua com a internet pode ser um problema em alguns cenários. Em geral, as ferramentas são acompanhadas por um conjunto de tutoriais com a intenção de apoiar o estudante nos pontos importantes sobre *ML.* Porém nenhuma destas ferramentas compõem os 5 estágios para a criação de modelos de detecção de objetos como: análise de requisitos, pré-processamento de dados, transferência de aprendizado, avaliação de desempenho, predição, e exportação do modelo.

Observa-se que além dos ambientes encontrados também existem outras ferramentas visuais para desenvolvimento de modelos de *ML* como Orange (www.orangedatamining.com), Microsoft Lobe (www.lobe.ai) e Amazon Rekognition (www.aws.amazon.com/pt/rekognition), etc. voltadas para usuários profissionais que de forma mais completa fornecem funcionalidades para diversas tarefas. Porém, como não foram encontrados relatos da aplicação destas ferramentas no ensino superior, essas não foram incluídas nesta revisão. Mesmo assim, observa-se o

potencial destas ferramentas para também suportar o ensino neste nível educacional.

Ameaças à validade. A fim de minimizar as ameaças à validade dos resultados deste estudo, foram identificadas ameaças potenciais e foram aplicadas estratégias de mitigação. As revisões sistemáticas podem ter viés comum de que resultados positivos têm maior probabilidade de serem publicados do que resultados negativos. No entanto, não consideramos isso uma ameaça crítica à esta pesquisa, pois em vez de focar no impacto dessas ferramentas, foi buscado caracterizar as próprias ferramentas. Para mitigar a omissão de estudos relevantes, foi construída cuidadosamente a *string* de busca para ser o mais inclusiva possível, considerando não apenas os conceitos principais, mas também os sinônimos. O risco de excluir estudos primários relevantes foi ainda mais atenuado pelo uso de vários bancos de dados e a inclusão de literatura secundária. Ameaças à seleção de estudos e extração de dados foram mitigadas por meio do fornecimento de uma definição detalhada dos critérios de inclusão/exclusão.

4. ODIN - camada visual para o desenvolvimento de modelos de detecção de objetos

Este capítulo apresenta ODIN uma camada de interface visual para o ensino de machine learning voltada a detecção de objetos no ensino superior.

4.1. Análise de contexto

O principal objetivo da unidade instrucional é ensinar *ML* no seguinte contexto: alunos iniciantes do ensino superior principalmente no nível de graduação em cursos de computação. Assume-se que os alunos tenham um conhecimento básico de programação com linguagens textuais, como Python.

Porém, tipicamente os alunos não possuem nenhum conhecimento de *IA* nem de tarefas básicas de *ML*, como detecção de objetos. Assim, necessitam inicialmente passar por uma etapa de aprendizagem mais conceitual de conceitos de *IA*/*ML* para um processo de aprendizagem adequado.

Em relação à infraestrutura, segundo dados do diagnóstico institucional da UFSC (2020) 93,18% dos alunos têm acesso a um computador de mesa ou notebook com acesso a internet. Porém, parte-se da premissa, que seus dispositivos podem não ser adequados para execução/treinamento das redes de DL. Sendo assim é importante que a execução não seja no computador do usuário, mas por exemplo por um servidor online gratuito, como Google Colab, o qual suportaria as atividades de desenvolvimento de *ML* pela alta demanda de processamento computacional e Google Drive para armazenamento de imagens.

4.2. Curso de ensino de *ML* para iniciantes no ensino superior

Atualmente está sendo desenvolvido um curso de *ML* voltado à tarefa de detecção de objetos para iniciantes no ensino médio e superior no contexto da iniciativa Computação na Escola/INCoD/INE/UFSC (Martins, 2022). Este curso visa ensinar o reconhecimento, compreensão e aplicação de conceitos de *ML* voltado a detecção de objetos focando no estágio *use* (do ciclo use-modifique-crie (Lytle et al., 2019)). Como trabalho prático, os estudantes neste curso irão aprender a construir um modelo para detecção de objetos de casa (cadeira, porta, armário, etc.) que será

implantado num aplicativo móvel que ajude as pessoas cegas ou deficientes visuais a identificar obstáculos no caminho delas no dia a dia.

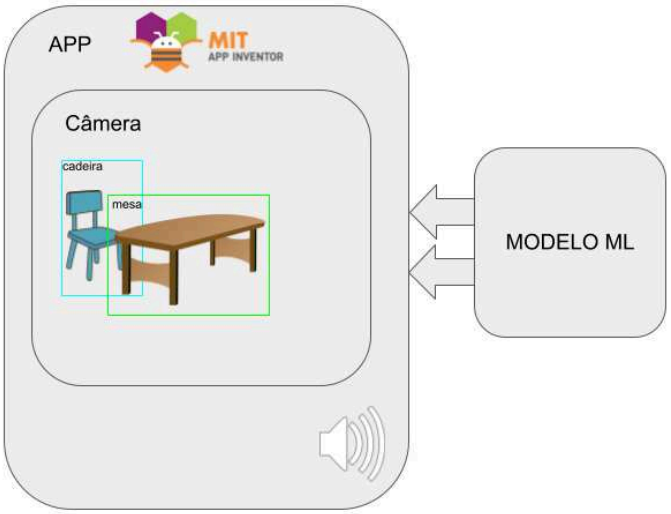


Figura 18 - Visão geral da integração do modelo de *ML* no Aplicativo Móvel

Ao final, o modelo de detecção de objetos será implantado num aplicativo móvel. Este aplicativo será implementado no App Inventor (https://appinventor.mit.edu/), um ambiente de programação visual baseado em blocos que permite rapidamente criar aplicativos móveis (Android e iOS) (MIT, 2019). A facilidade de uso do App Inventor o torna uma poderosa ferramenta tanto para o ensino de programação quanto para a realização de projetos de aplicativos por parte de usuários finais, profissionais de diversas áreas, sem necessidade de familiaridade com linguagens de programação e ferramentas mais avançadas (MIT, 2019).

Capturando cenas de ambientes internos via a câmera do celular, foi executado o modelo de *ML* no aplicativo para detectar objetos de casa e avisando o usuário da existência dos objetos na sua frente, transformando o resultado da detecção de objetos em forma de texto em saída auditiva. Com base nos resultados da detecção de objetos, estes avisos auditivos incluem tanto o nome do objeto (p.ex., cadeira) como uma indicação da posição dos objetos (à direita, esquerda, etc.) que sinalize objetos identificados próximos pelo modelo de *ML* por meio da câmera do aplicativo.

Dentro dos objetivos de aprendizagem previstos nesse curso o aluno deve aprender todo o processo iterativo de desenvolvimento de modelo de *ML* desde a análise de requisitos até a implantação no App Inventor. O curso deve cobrir todo o processo na criação do modelo de *ML*, como análise de requisitos, preparação de dados, treinamento e avaliação do modelo gerado, exportação (*ONNX*) e a implantação. Esses objetivos serão alcançados com a tarefa de detecção de objetos, com o intuito de ser aplicado para desenvolver um app para cegos.

O desenvolvimento do modelo de *ML* será realizado no ambiente Jupyter/Colab por ser um ambiente já mais convencional, pela facilidade de gratuitamente armazenar os dados (*Google Drive*) e via *colab* usar os recursos do *Google* na nuvem para treinar gratuitamente os modelos. Visando a implantação em aplicativo móvel será disponibilizada arquitetura leve como *YOLOv5s*, que utiliza uma rede neural mais simples visando menor tempo de detecção e peso computacional, o que torna esta versão do *YOLO* ideal para aplicações de vídeo em tempo real.

Assim, considerando o nível iniciante neste curso, visa-se num primeiro momento usar uma interface visual para facilitar a aprendizagem conceitual de ML.

4.3. Análise de requisitos

Considerando o objetivo de facilitar o aprendizado dos conceitos de ML no início da disciplina, visa-se criar uma camada visual para possibilitar a criação de modelos de *ML* de forma prática, levando o aluno ao nível de aplicação destes conceitos. Prevê-se a criação de uma camada visual sobre um Jupyter notebook para facilitar a transição posterior para um ambiente textual. Levando em consideração que já existem primeiras soluções deste tipo de camada visual para Jupyter (Franz, 2021), será criado um ambiente visual para a tarefa de detecção de objetos. Visa-se utilizar *ipywidgets* (www.ipywidgets.readthedocs.io) e adaptando a interface visual criada por Franz (2021) voltada a apoiar o desenvolvimento de modelo de *ML* para tarefa de classificação de imagens.

O suporte visual para criação de modelos de *ML* para detecção de objetos será instanciado pela criação exemplar de um modelo de *ML* para detecção de

objetos de casa (cadeira, porta, sapato, mesa, etc.) em termos de localização (na frente, a direita, a esquerda) visando ser um suporte para um app para ajudar pessoas com alguma deficiência visual.

A coleta de dados não será suportada pela ferramenta, prevê-se a realização do processo de rotulação/anotação das imagens via a ferramenta do *roboflow (https://roboflow.com/)* ao final o *dataset* é exportado no formato *YOLOv5 Pytorch.*

O suporte será integrado no *Jupyter Notebooks* por ser um dos ambientes textuais tipicamente adotados no ensino superior para o desenvolvimento de modelos de *ML*. Integrando o suporte diretamente no Jupyter Notebooks permitirá uma transição direta e rápida do estudante do ambiente visual para o textual no decorrer de sua progressão, reduzindo o esforço para a aprendizagem em relação ao uso das ferramentas de *ML*.

Observa-se a possibilidade de utilizar uma infraestrutura online e gratuita pelo *Google Colab* ou outra ferramenta, para que não haja dependência de computadores com uma capacidade computacional compatível com o *ML*, e para também mitigar problemas com ambiente como sistema operacional, linguagens e bibliotecas, será montado sobre o *Jupyter Notebooks/Google Colab* para facilitar e possibilitar a massiva utilização da ferramenta.

O layer visual deve permitir:

- Obtenção de imagem do *Google drive*.
- Obtenção do conjunto de dados anotados pela ferramenta *Roboflow (https://roboflow.com/*)) armazenado no Google Drive.
- Visualização das imagens do conjunto de dados (objetos por categoria, estatísticas, como a distribuição de objetos por imagens), etc.
- Treinamento de modelos *ML* selecionado o modelo, e hiperparâmetros como por exemplo *batches*, *image size* e *epochs*.
- Visualização de medidas utilizadas para analisar o desempenho referente a tarefa de detecção de objetos como, por exemplo, mean *Average Precision, precision, recall, entre outros.*

- Possibilidade de *upload* ou captura de imagens novos para testes da predição.
- Salvar o modelo treinado.
- Exportar o modelo usando o padrão *ONNX, TorchScript, Tensorflow Lite, pt, pb*.

Requisitos não funcionais:

- O ambiente precisa estar dentro do *Jupyter Notebook*.
- Treinamento precisa ser executável no ambiente online (*Google Colab*) com menos do que 2 horas (considerando um conjunto de dados com pelo menos 250 imagens)

A implantação do modelo em um app no App Inventor está fora do escopo do presente trabalho. Porém ao final do desenvolvimento, a interface será capaz de ser aplicada, por ser uma ferramenta genérica e por exportar em padrão suportado pela maioria das ferramentas de *ML*.

4.4. Arquitetura do sistema

A arquitetura apresentada na Figura 18 consiste em um ambiente que rode sobre um *Jupyter Notebooks*, uma vez que um dos objetivos deste trabalho popularizar e democratizar o acesso a *IA*/*ML* e os *Jupyter Notebooks* são acessíveis diretamente pelo navegador do computador, independentes do sistema operacional. Como principal opção para viabilizar o uso deste ambiente tem-se o *Google Colab*, que é possível utilizar a plataforma de forma gratuita, sem a necessidade de instalação de programas e/ou bibliotecas adicionais, além de disponibilizar GPUs. Sobre o Colab será montado um ambiente de programação visual em que será possível realizar todas as tarefas relacionadas a ML foi utilizada a biblioteca *Jupyter Widgets* (ipywidgets, 2021) que é a principal referência em elementos visuais utilizados em *Jupyter Notebooks*. Para a tarefa de detecção de objetos é utilizado a biblioteca *PyTorch* que é utilizada principalmente para o desenvolvimento de modelos de detecção de objetos YOLOv5, que tem total integração com o Google Colab.

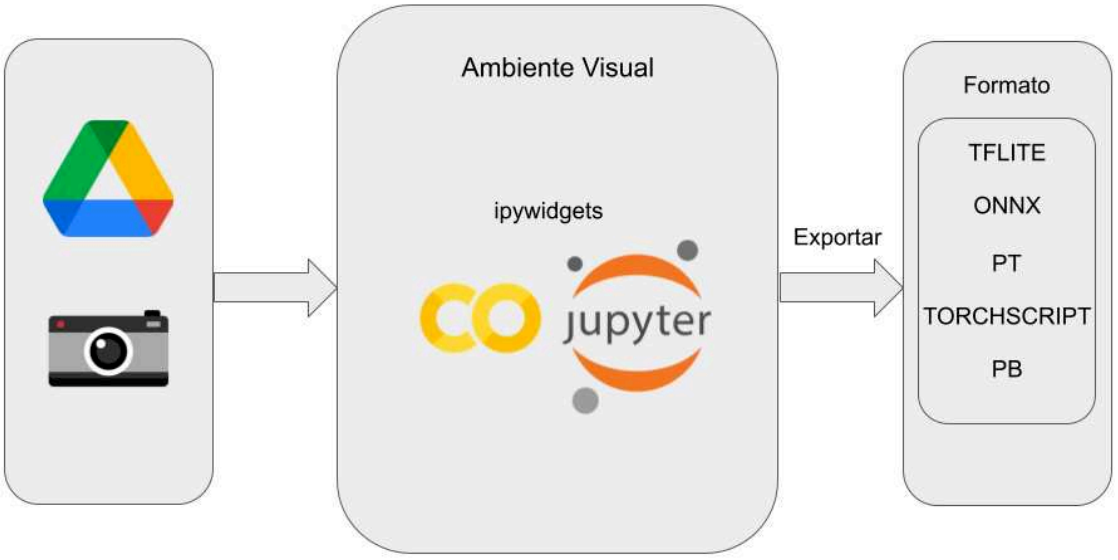


Figura 19 - Visão geral da arquitetura do sistema

Apesar de suportar elementos visuais, o Colab não foi concebido como uma interface gráfica ou aplicação de uma célula. Normalmente se utilizam diversas células, uma com cada trecho de código referente a etapa do processo de ML. Entretanto é possível encapsular todos os processos de IA/ML de acordo com os elementos visuais disponíveis e executar a aplicação de ponta a ponta em apenas uma célula, escondendo em si o código da aplicação como demonstrado na Figura 19.

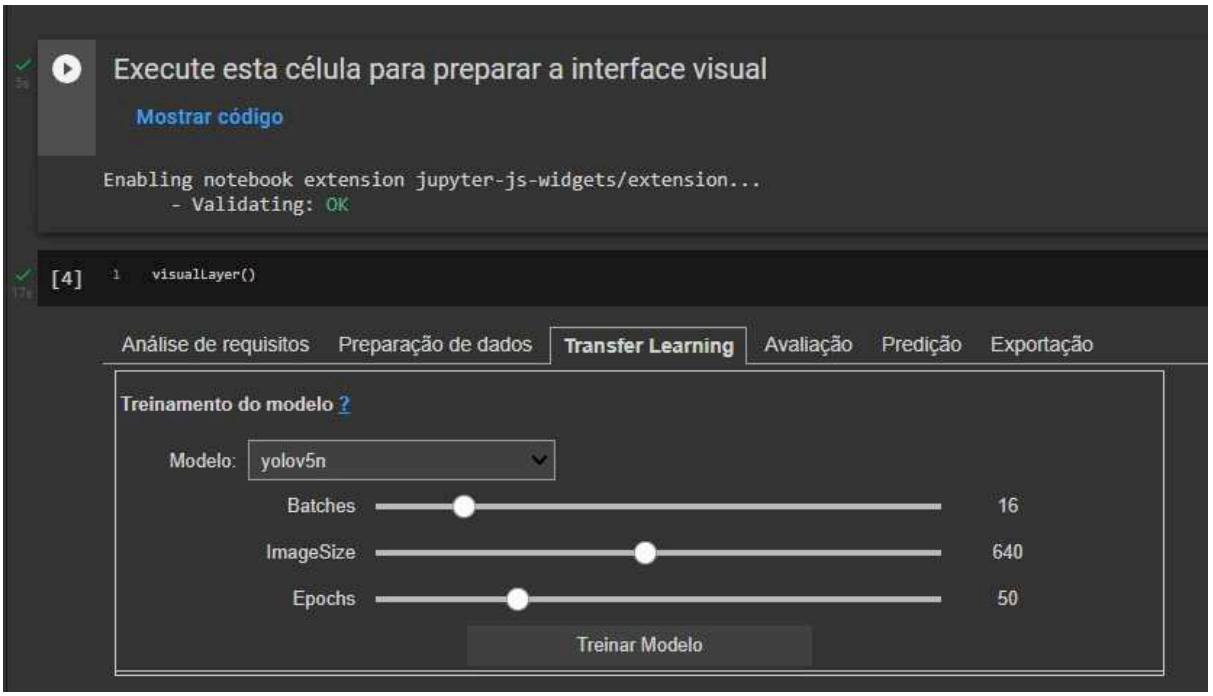


Figura 20 - Interface do ODIN

4.5. Implementação do ODIN

Nesta seção é apresentado o desenvolvimento e implantação da biblioteca. Toda a estrutura proposta foi desenvolvida utilizando Python (Python, 2021) por rodar nativamente no colab e ter total compatibilidade com os frameworks selecionados. De acordo com a análise de requisitos e a arquitetura definida seguindo o human-centric interactive ML process (Gresse Von Wangenheim e Von Wangenheim, 2021) foram especificadas as necessidades de entrada e saída de dados guiando o suporte desenvolvido *ODIN* conforme apresentado na Tabela 13.

Tabela 13 - Definição do suporte do ODIN

Fase do Processo	**Passo do Processo**	**Detalhes e Padrões**
Análise de Requisitos	Especificação da(s) tarefa(s) e requisitos do sistema de ML	● Tarefa ● Tipo da tarefa ● Categorias ● Fonte de dados ● Quantidade de dados ● Padronização das imagens ● Rotulação dos dados ● Desempenho
Preparação do Dados	Conjunto de dados anotado (no Google Drive)	Um conjunto de dados de imagens já anotado (com formato para YOLOV5) deve ser disponibilizado em uma pasta no Google Drive e dentro desta pasta deve haver 3 pastas e um arquivo .yaml: uma pasta para treinamento, uma para validação e outra para teste, e um arquivo de configuração dentro de cada respectiva pasta é preciso ter 2 pastas, uma para as imagens e a outra para os labels.
	Definição do conjunto de dados	Caminho do diretório no Google Drive
	Visualizar informações do conjunto de dados	Percorre o conjunto de dados total informado e mostra um histograma com a distribuição dos objetos por classe
	Visualizar conjunto de dados	Percorre o conjunto de dados de treinamento informado e mostra até 6 exemplos com suas respectivas labels
Transfer Learning	Seleção do modelo	Padrão = yolov5n ● yolov5n ● yolov5s ● yolov5m ● yolov5l ● yolov5x
	Tamanho do batch	Padrão = 16 ● min = 1 ● max = 128
	Tamanho das imagens	Padrão = 640 ● min = 64 ● max = 1280
	Quantidade de épocas	Padrão = 50 ● min = 1 ● max = 500
	Treinar modelo	Executa o treinamento do modelo escolhido a partir da quantidade de épocas, tamanho dos batches e tamanho de imagem informada.
Avaliação	Avaliação do treinamento	Exibe as principais métricas para validar o treinamento do modelo: ● train/box_loss ● train/obj_loss ● train/cls_loss ● metrics/precision ● metrics/recall ● val/box_loss ● val/obj_loss

		• val/cls_loss • metrics/mAP_0.5 • metrics/mAP_0.5:0.95
	Curva do F1 score	Exibe o F1 Score
	Matriz de confusão	Exibe a matriz de confusão
Predição	Taxa de confiança	Padrão = 10 • min = 1 • max = 100
	Predição pelo conjunto de teste	Faz a predição pelas imagens da pasta test do Dataset
	Predição pelo *upload*	Permite o upload pelo HD de uma imagem (por vez) para realizar a predição da mesma.
Exportação	Seleção de formato	Padrão = pt • tflite • torchscript • pb • onnx • pt
	Definição do caminho	Caminho do diretório no Google Drive para exportar o modelo no formato
	Exportar	Executa o algoritmo para transformar para o formato escolhido e cópia para a pasta definida no passo anterior

4.5.1. Integração com o Google Drive

A integração com o Google Drive (Colab, 2021) é a ponte entre a entrada e saída dos dados com o *ODIN*, permitindo tanto a entrada do conjunto de dados para o treinamento do modelo quanto a exportação do modelo treinado.

A autenticação é feita com a API da Google e é possível logar com a conta que desejar (não necessariamente a mesma do Colab), esta conta se conectará ao drive apenas nas pastas informadas pelo próprio usuário durante a execução.

Para integrar o conjunto de dados deve conter 3 pastas, uma para treinamento, uma validação e outra para teste e um arquivo *.yaml* de configuração.

4.5.2. Biblioteca Ultralytics/yolov5

Utilizada uma série de arquivos da biblioteca do Ultralytics/yolov5 (github.com/ultralytics/yolov5) para realizar o processo completo de ML. São utilizados os seguintes objetos da biblioteca:

- train.py é o arquivo utilizado para treinar o modelo, as informações utilizadas para execução do treino são passadas por parâmetro como batches, imageSize, epochs entre outros;
- detect.py é o arquivo utilizado para detectar objetos em uma imagem, as informações utilizadas na execução são passadas por parâmetro, como taxa de confiança, modelo, entre outros.
- val.py é o arquivo utilizado para avaliar o modelo treinado retornando o mAP de geral e para cada classe do conjunto de dados.

4.5.3. Processo de ML

O fluxo do processo de ML no *Visual Layer* foi desenvolvido seguindo o processo de desenvolvimento de ML centrado no ser humano conforme apresentado na Figura 20 (Gresse von Wangenheim e von Wangenheim, 2021), e também nas necessidades da biblioteca do *Ultralytics/yolov5* apresentadas na seção 2.1. Para desenvolver essa camada intermediária foram criadas algumas funções para garantir os objetos necessários além da execução das etapas pré-definidas.

Figura 21 - Etapas do processo de ML centrado no ser humano

create_config_files(): Responsável pela atualização do arquivo *data.yaml,* alterando o caminho definido no arquivo para as pastas *train* e *valid* do dataset definido pelo usuário;

charge_yolov5(): Responsável pela importação do projeto do *YOLOv5* e instalação de seus pacotes requeridos;

drive_connect(): Responsável pela abertura de conexão com o Google Drive e definição e gravação do caminho raiz na variável *root_dir*.

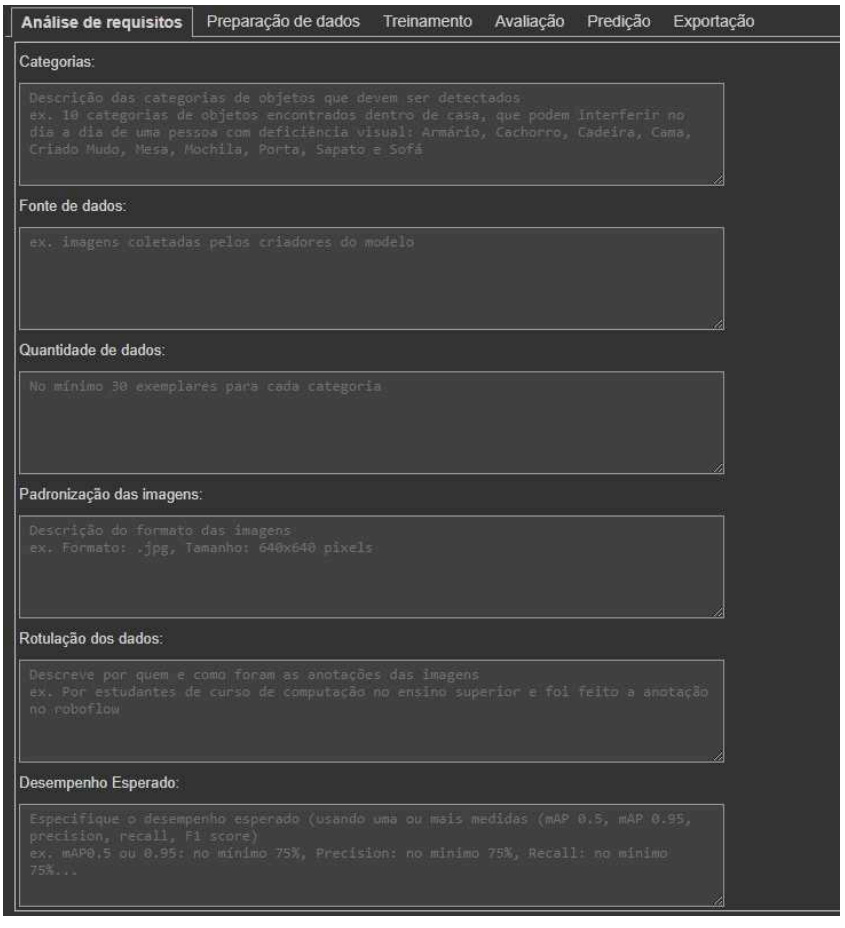
render_odin(): Responsável pela renderização de todos os componentes e funções intermediárias como charge_yolov5() e drive_connect();

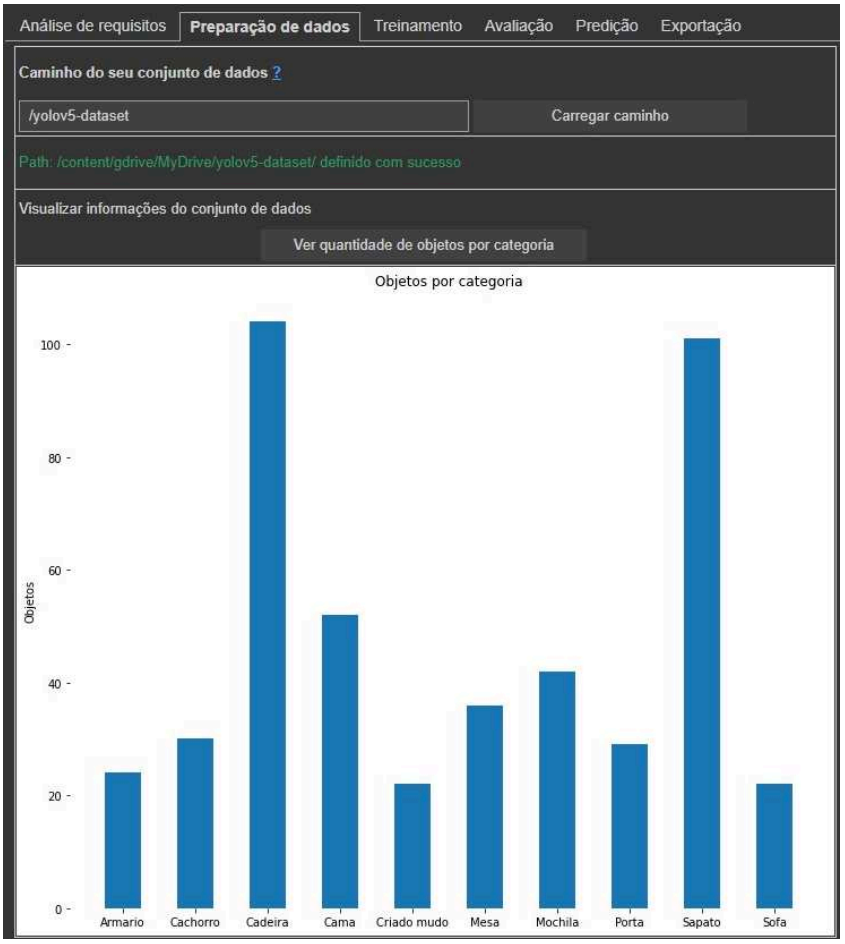
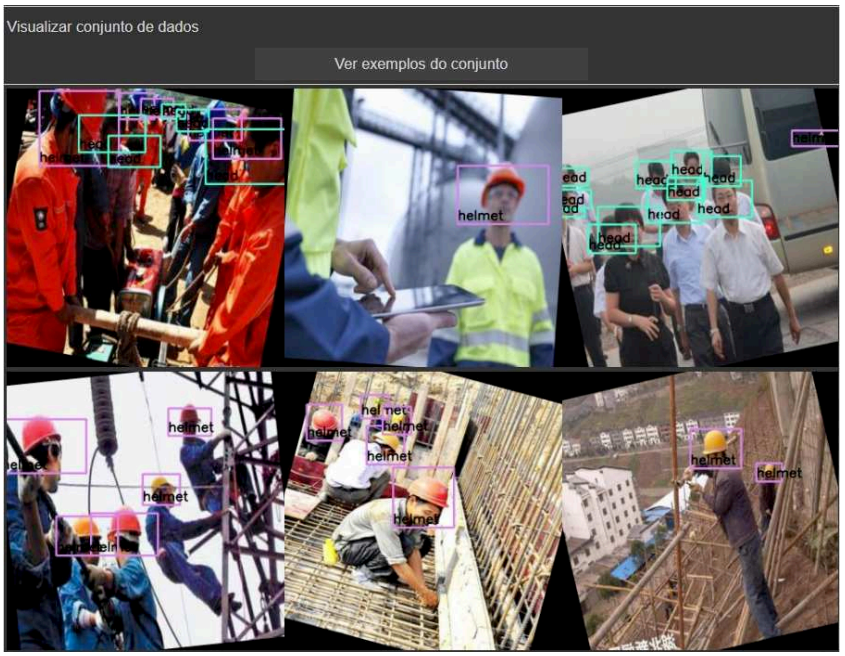
4.5.4. Interface visual

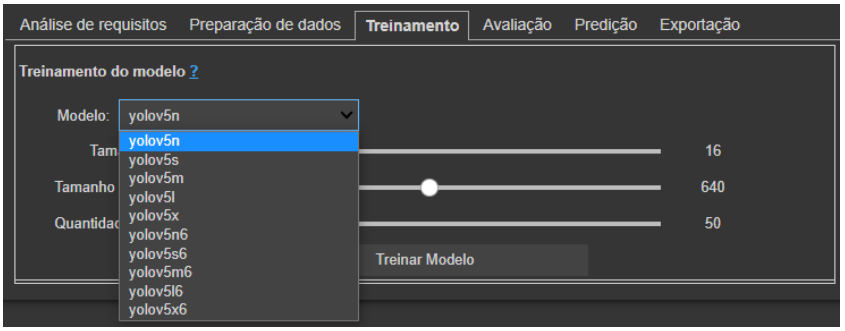
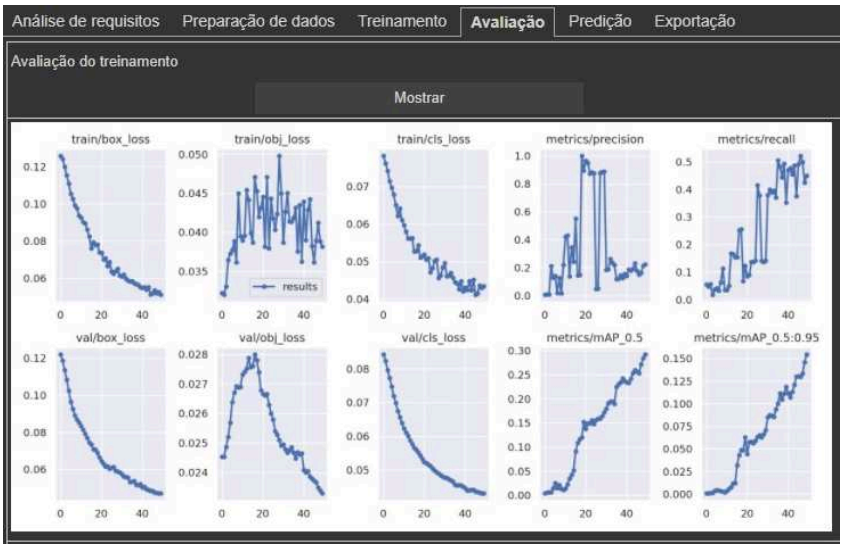
A interface representa visualmente todo o processo de ML/IA na ferramenta, integrando todos os elementos. Para criar um ambiente sem código, tudo foi encapsulado na função *render_odin()*.

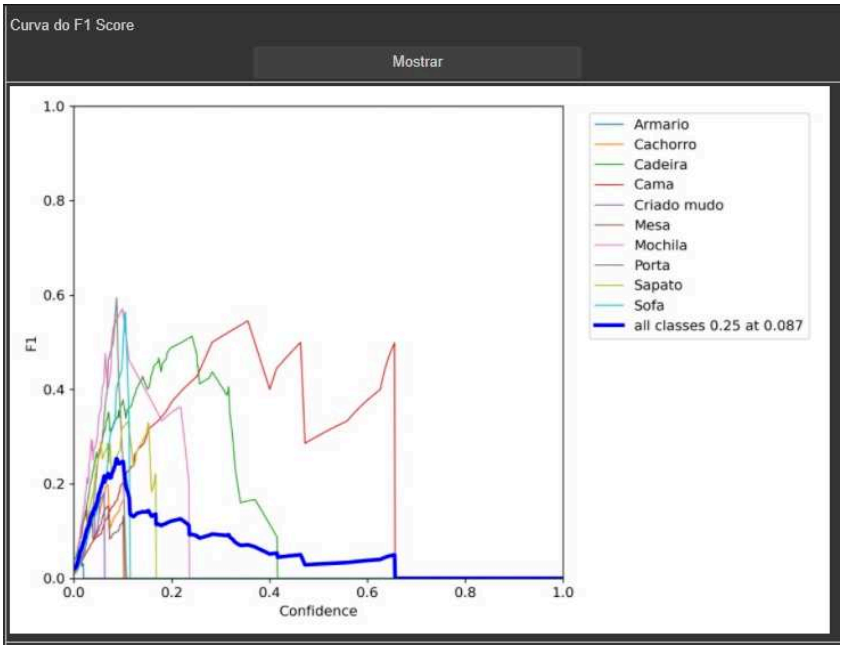
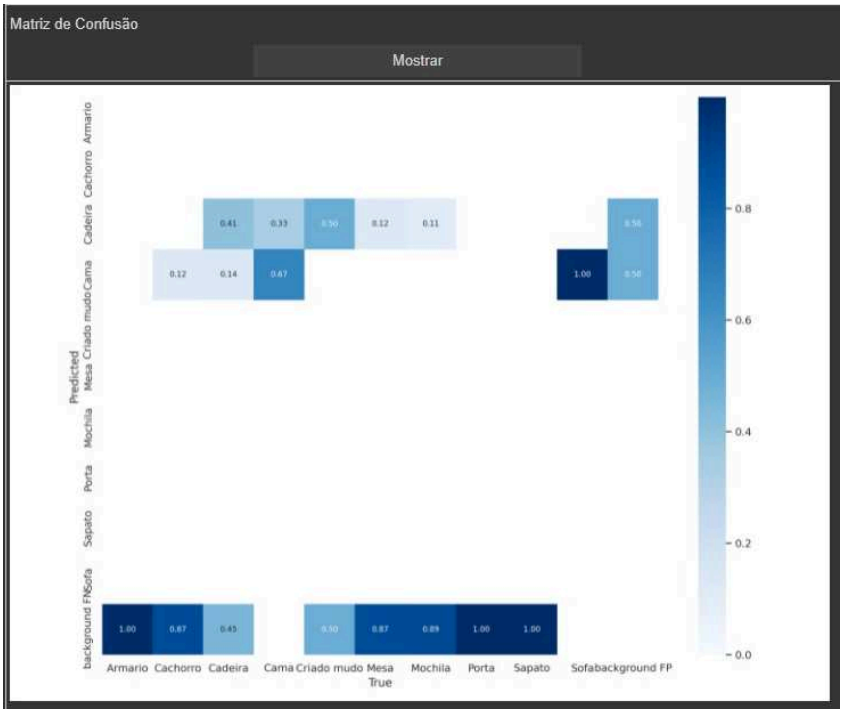
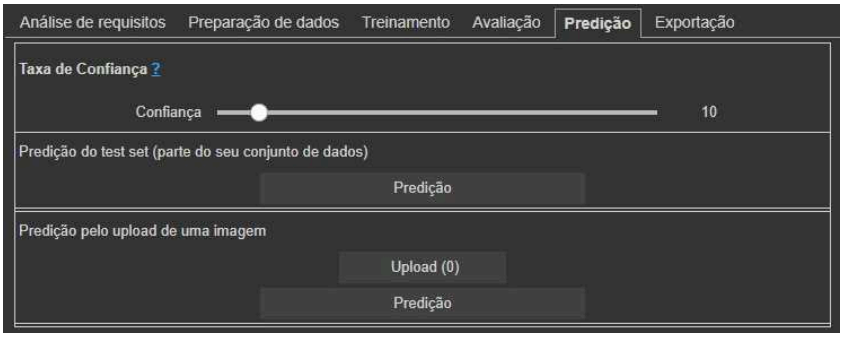
A base interface visual da ferramenta foi criada com a biblioteca do Google Colab widgets. Foram utilizados tabs para criação das abas e grids para conter os visuais e saídas da ferramenta. Para os demais elementos (textos, entradas e botões) foi utilizado a biblioteca ipywidgets com seus diversos *widgets*, *styles*, *layouts* e interações.A Tabela 14 mostra o resultado da interface visual desenvolvida (https://codigos.ufsc.br/gqs/odin).

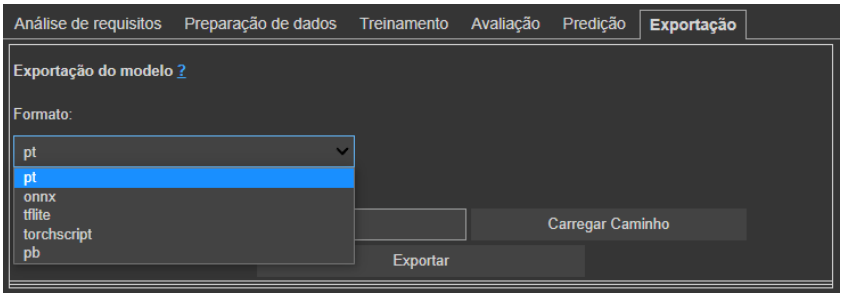
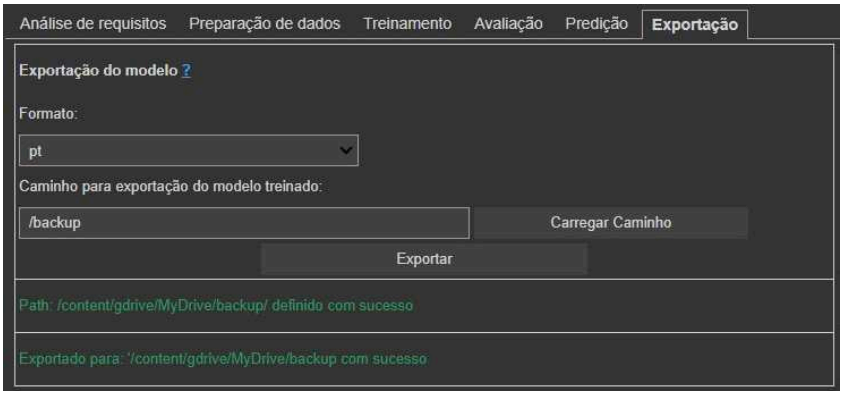
Tabela 14 - Interface visual do ODIN

Fase do Processo	**Passo do Processo**	**Interface visual**
Análise de Requisitos	Especificação da(s) tarefa(s) e requisitos do sistema de ML	

Preparação dos Dados	Conjunto de dados (no Google Drive)	
	Definição do conjunto de dados	
	Visualizar informações do conjunto de dados	
	Visualizar conjunto de dados	

<table>
<tr><td rowspan="4">Transfer Learning</td><td>Seleção do Modelo</td><td rowspan="2"> 
</td></tr>
<tr><td>Tamanho do batch
Tamanho das imagens
Quantidade de épocas
Treinar modelo</td></tr>
<tr><td rowspan="2">Avaliação do desempenho</td><td>Avaliação do treinamento</td><td> 
</td></tr>
</table>

	Curva do F1 Score	
	Matriz de confusão	
Predição	Taxa de confiança	
	Predição pelo conjunto de teste	
	Predição pelo *upload*	

<table>
<tr><td rowspan="3">Exportação</td><td>Seleção do formato</td><td rowspan="2"> 
</td></tr>
<tr><td>Definição do caminho</td></tr>
<tr><td>Exportar</td><td> 
</td></tr>
</table>

5. Conclusão

Neste trabalho de conclusão de curso, foram apresentados os conceitos de *machine learning* e buscadas ferramentas visuais com o propósito de instruir estudantes do ensino superior a iniciarem seu desenvolvimento na área de *machine learning* como no caso da ferramenta do *Google Teachable Machine*.

Com essa base de conhecimento foi implementada uma camada visual a qual suporta o processo de desenvolvimento de sistemas de *Machine Learning* no contexto do ensino superior, a ferramenta ODIN é capaz de criar modelos sem a necessidade de código textual por meio de 6 fases para o desenvolvimento: análise de requisitos, preparação dos dados, treinamento, avaliação, predição e exportação. O ODIN é uma ferramenta prática que abstrai as dificuldades da configuração do projeto e organiza o processo da criação até a exportação do modelo. Para trabalhos futuros planeja-se a adição do passo de *fine tuning* (ajuste fino), a inclusão de mais hiperparâmetros para customizar o treinamento, *augmentation* para equilibrar o dataset, realização de uma avaliação do desempenho da ferramenta e a integração a um curso no ensino superior.

REFERÊNCIAS

ACM, Computer Science Curricula, 2013. Disponível em: <https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf> Acesso em Maio de 2020.

S. Amershi, et al. 2019. Software Engineering for Machine Learning: A Case Study. In Proc. of the 41st International Conference on Software Engineering: Software Engineering in Practice, IEEE Press, 291–300.

Amazon. 2019. *Amazon Machine Learning*, AWS Documentation. https://docs.aws.amazon.com/machine-learning/latest/dg/building-machine-learning.html.

M. Armoni, O. Meerbaum-Salant, e M. Ben-Ari. 2015. From Scratch to “Real” Programming.

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

V. Bushaev ; How do we ‘train’ neural networks?, 2017. Disponível em:<https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94> . Acesso em Agosto de 2020.

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

M. Carney, B. Webster, I. Alvarado, K. Phillips, N. Howell, J. Griffith, J. Jongejan, A. Pitaru, and A. Chen. 2020. Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification. In Proc. of Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems. ACM, New York, NY, USA, 1–8.

I. Costa , E. Silva Jr , A. Rodrigues , L. Angeloni e E. Dias. 2020. Avaliação do Processo para Embarcar uma Rede Neural Baseada em YOLO Utilizando um Acelerador de Hardware Dedicado.

M. Copeland. What 's the Difference Between Artificial Intelligence, Machine Learning and Deep Learning? Disponivel em: <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/> Acesso em Maio de 2020.

P. Dorotea, Nuno & Sampaio, Fábio & Pedro, Ana. (2019). A Cross-analysis of Block-based and Visual Programming Apps with Computer Science Student-Teachers. Education Sciences, 9, 181.

J. Dudley, P. Kristensson 2018. A Review of User Interface Design for Interactive Machine Learning.ACM Trans. Interact. Intell. Syst,8(8), 2018.

Fiebrink, Rebecca & Gillies, Marco. (2018). Introduction to the Special Issue on Human-Centered Machine Learning. ACM Transactions on Interactive Intelligent Systems. 8. 1-7.

C. Gresse von Wangenheim, J. Hauck, F. Pacheco, et al. Visual tools for teaching machine learning in K-12: A ten-year systematic mapping. Educ Inf Technol 26, 5733–5778 2021.

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

M. Gillies, et al. (2016) Human-Centered Machine Learning. Proc. of the Conference on Human Factors in Computing Systems, ACM, San Jose, CA, USA, 3558-3565, https://hal.inria.fr/hal-01437057/document.

L. Goasduff. This Gartner Hype Cycle highlights how AI is reaching organizations in many different ways, 2019.Disponivel em <https://www.gartner.com/smarterwithgartner/top-trends-on-the-gartner-hype-cycle-for-artificial-intelligence-2019/> Acesso em Maio de 2020

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

W. Hechun and Z. Xiaohong. Survey of Deep Learning Based Object Detection. In Proceedings of the 2nd International Conference on Big Data Technologies. ACM, New York, NY, USA, 2019., 149–153.

A. Hoover, A. Spryszynski, and M. Halper. 2019. Deep Learning in the IT Curriculum. In Proc. of the 20th Annual SIG Conference on Information Technology Education . ACM, New York, NY, USA, 49–54.

D. Hils, Visual languages and computing survey: Data flow visual programming languages. 1992. J. Vis. Lang. Comput., vol. 3, no. 1, pp. 69–101, Mar.

C. Hmelo e M. Guzdial. Of black and glass boxes: scaffolding for doing and learning. In Proc. of the international Conference on Learning sciences. International Society of the Learning Sciences, 1996 , 128–134.

L. Jiao et al. A Survey of Deep Learning-Based Object Detection. IEEE Access, 7, 128837-128868, 2019.

J. Johnson. 2019. Teaching neural networks in the deep learning era. J. Comput. Sci. Coll., 34(6) 2019, 16–25.

J. Johnson. 2020. Benefits and Pitfalls of Jupyter Notebooks in the Classroom. In Proceedings of the 21st Annual Conference on Information Technology Education. Association for Computing Machinery, New York, NY, USA, 32–37.

W. Johnston, J. Hanna, R. Millar Advances in Dataflow Programming Languages. 2004. ACM Comput. Surv., vol. 36, no. 1, pp. 1–34, Mar.

T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, et al. 2016. Jupyter notebooks – a publishing format for reproducible computational workflows. (Loizides F& Schmidt B, Eds.).

J. Zhang, P. Qu, C. Sun et al., “DWCA-YOLOv5: An improve single shot detector for safety helmet detection,” *Journal of Sensors*, 2021

D. Knuth, L. Pardo, 1980. The early development of programming languages. In A history of computing in the twentieth century, 197–273.

K. Kahn, N. Winters, Child-Friendly Programming Interfaces to AI Cloud Services. 2017. In: Lavoué É. et al. (eds) Data Driven Approaches in Digital Education. Lecture Notes in Computer Science, vol 10474. Springer, Cham.

P. Karkare; Convolutional Neural Networks – Simplified, 2019. Acesso em: Agosto de 2021

B. Kenji, 2019; Machine Learning para Leigos. Disponível em: Machine Learning para Leigos | Venturus. Acesso em: Julho de 2021.

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

Y. Lecun, Y. Bengio, G. Hinton. Deep Learning, 2015. Nature, 521, 436-44.

MIT App Inventor. About Us, 2019. Disponível em: About Us (mit.edu) Acesso em: Julho de 2021.

R. Meulen e C. Pettey, Gartner Says by 2020, Artificial Intelligence Will Create More Jobs Than It Eliminates.Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2017-12-13-gartner-says-by-2020-artificial-intelligence-will-create-more-jobs-than-it-eliminates > Acesso em Maio de 2020

K. Mathewson, (2019). A Human-Centered Approach to Interactive Machine Learning. arXiv:1905.06289v1 [cs.HC].

M. Nelson e A. Hoover. 2020. Notes on Using Google Colaboratory in AI Education. In Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20). Association for Computing Machinery, New York, NY, USA, 533–534.

G. Nodalo, J. Santiago, J. Valenzuela, and J. Aiko Deja. 2019. On Building Design Guidelines for An Interactive Machine Learning Sandbox Application. In Proc. of the 5th International ACM In-Cooperation HCI and UX Conference,ACM, New York, NY, USA, 70–77.

J. Perkel. 2018. Why Jupyter is data scientists' computational notebook of choice. Nature 2018.

J. Piazentin Ono, J. Freire and C. Silva, "Interactive Data Visualization in Jupyter Notebooks," in Computing in Science & Engineering, vol. 23, no. 2, pp. 99-106, 1 Mar 2021.

E. Pasternak, R. Fenichel and A. N. Marshall.Tips for creating a block language with blockly," 2017 IEEE Blocks and Beyond Workshop, Raleigh, NC, 2017, 21-24.

Pathmind, A Beginner's Guide to Neural Networks and Deep Learning. Disponível em:<https://pathmind.com/wiki/neural-network>. Acesso em Julho de 2020a.

Pathmind, A Beginner's Guide to LSTMs and Recurrent Neural Networks. Disponível em:<https://pathmind.com/wiki/lstm>. Acesso em Julho de 2020.

Pathmind, A Beginner's Guide to Convolutional Neural Networks (CNNs). Disponível em:<https://pathmind.com/wiki/convolutional-network>. Acesso em Julho de 2020c.

Pant, A. Workflow of a Machine Learning Project, 2019. Disponível em: <https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94?gi=ef53a0d0e22>. Acesso em Agosto de 2020

R. Padilla, S. Netto e E. da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms," 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), 2020.

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

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

Robinson, 2018. Sketch2code: Generating a website from a paper mockup. Dissertation, University of Bristol, UK.

G. Ramos, C. Meek, P. Simard, J. Suh, S. Ghorashi. (2020) Interactive machine teaching: a human-centered approach to building machine-learned models, *Human–Computer Interaction*, 35(5-6).https://www.microsoft.com/en-us/research/uploads/prod/2020/05/Interactive_Machine_Teaching__Free_access_.pdf

G. Ramos, C. Meek, P. Simard, J. Suh, S. Ghorashi. (2020) Interactive machine teaching: a human-centered approach to building machine-learned models, *Human–Computer Interaction*, 35(5-6).https://www.microsoft.com/en-us/research/uploads/prod/2020/05/Interactive_Machine_Teaching__Free_access_.pdf

S. Russell e P. Norvig Artificial Intelligence A Modern Approach: Englewood Cliffs, New Jersey: Prentice-Hall, 1995.

R. Sinnott, D. Yang, X. Ding, and Z. Ye. 2020. Poisonous Spider Recognition through Deep Learning. In Proc. of the Australasian Computer Science Week Multiconference. Association for Computing Machinery, New York, NY, USA, Article 14, 1–7

S. Tamilselvam, N. Panwar, S. Khare, R. Aralikatte, A. Sankaran, S. Mani. A visual programming paradigm for abstract deep learning model development. In Proc. of the 10th Indian Conference on Human-Computer Interaction. ACM, New York, NY, USA, 2019. Article 16, 1–11.

UFSC, Diagnóstico Institucional UFSC Junho 2020. Disponível em: 15_06_20_DIAGNOSTICO_INSTITUCIONAL_1.pdf (ufsc.br). Acesso em Junho de 2021.

A. Wangenheim. Detecção de objetos em imagens. 2018 Disponível em: Deep Learning::Detecção de Objetos em Imagens (ufsc.br). Acesso em 2021 de Agosto.

Wissal Farsal, Samir Anter, and Mohammed Ramdani. 2018. Deep Learning: An Overview. In Proceedings of the 12th International Conference on Intelligent Systems: Theories and Applications (SITA'18). ACM, New York, NY, USA, Article 38, 1–6.

D. Weintrop, W. Uri.(2017). How block-based languages support novices: A Framework for categorizing block-based affordances. Journal of Visual Languages and Sentient Systems, 3. 92-100.

D. Weintrop, H. Killen, T. Munzar, and B. Franke. 2019. Block-based Comprehension: Exploring and Explaining Student Outcomes from a Read-only Block-based Exam. In Proc. of the 50th ACM Technical Symposium on Computer Science Education. Association for Computing Machinery, New York, NY, USA, 1218–1224.

Y. Watanabe, et al. (2019). *Preliminary Systematic Literature Review of Machine Learning System Development Process*. arXiv:1910.05528 [cs.LG].

D. Weintrop. Block-based Programming in Computer Science Education". Communications of the ACM, 62(8), 2019, 22-25.

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

APÊNDICE A

ODIN - Uma Ferramenta Visual para o Desenvolvimento de Modelos de Detecção de Objetos com *Deep Learning* no Ensino Superior

João Pedro Santana
Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística
Florianópolis SC Brasil
joao.pedro.santana@grad.ufsc.br

Resumo. Nos últimos anos, projetos na área de *Machine Learning (ML)* e especialmente de *Deep Learning (DL)* estão presentes no dia a dia, em carros autônomos, antivírus e até no reconhecimento de padrões de imagens. Assim, o *Deep Learning* representa uma área de conhecimento importante na ciência da computação voltada a diversas tarefas como classificação de imagens ou detecção de objetos. Porém, sua complexidade pode ser uma barreira para os iniciantes, especialmente quando se utilizam ferramentas, arquiteturas e bibliotecas de contexto profissional. O trabalho inclui o desenvolvimento de uma camada visual para *Jupyter notebook* no *Google Colab*. Espera-se com isso ajudar os iniciantes a darem o primeiro passo para aprender conceitos básicos de *DL*, ao mesmo tempo preparando para uma fácil transição a ambientes convencionais usando linguagens de programação textuais.

Abstract. In recent years, projects in the area of Machine Learning (ML) and especially Deep Learning (DL) are present in everyday life, in autonomous cars, antivirus and even in image pattern recognition. Thus, Deep Learning represents an important area of knowledge in computer science aimed at various tasks such as image classification or object detection. However, its complexity can be a barrier for beginners, especially when using tools, architectures and libraries in a professional context. The work includes the development of a visual layer for Jupyter notebook in Google Colab. This is expected to help beginners take the first step towards learning basic DL concepts, while also preparing for an easy transition to conventional environments using textual programming languages.

PALAVRAS CHAVES
Machine Learning, *Deep Learning,* Ambiente visual, Detecção de objetos, Ensino superior, *Jupyter Notebook, Google Colab*

1. Introdução

A Inteligência Artificial (IA) tem feito parte da imaginação por muitos anos nos filmes de ficção científica apresentando máquinas que fazem muito mais do que deveriam, como uma inteligência a qual poderia pensar como os humanos. Porém, esta ideia fica só em filmes e livros, pelo fato de ainda não ser possível a concretização desse sonho com as tecnologias existentes. Com este limite pressuposto das tecnologias atuais de nossas tecnologias criou-se um novo conceito "*Narrow Artificial Intelligence*", tecnologias que são capazes de executar tarefas específicas tão bem quanto, ou até melhor, que humanos [Copeland 2016]. Com esta descoberta e mudança de percepção sobre a Inteligência Artificial criou-se o subcampo chamado de *Machine Learning (ML)* o qual tem o intuito de criar modelos que aprendam com experiências ou exemplos [Russel et al. 1995]. *ML* é a prática de usar algoritmos para coletar dados, aprender com eles, e então fazer uma determinação ou predição sobre alguma coisa no mundo.

Deep learning (DL) é um subcampo de *Machine Learning*, o qual usa uma rede neural que tem como base o aprendizado por técnicas de *ML* [Hoover et al. 2019]. O *DL* está cada vez mais presente, em produtos de consumo como celulares, computadores, câmeras e outros eletroeletrônicos, utilizado para combinar notícias, postagens ou produtos vinculados aos interesses dos usuários [Lecun et al. 2015]. O *DL* permite que modelos computacionais compostos por várias camadas de processamento, aprendam representações de dados com múltiplos níveis de abstrações [Lecun et al. 2015]. Estes métodos novos provenientes do *DL* melhoraram drasticamente o estado da arte no reconhecimento de fala, detecção de objetos e vários outros domínios em geral [Lecun et al. 2015].

Pela alta demanda por projetos na área da IA é estimada a criação de mais de 2 milhões de empregos na área em 2025 [Meulen et al. 2017], tornando-se uma das tecnologias que mais crescem no mundo da ciência da computação. Porém, esse crescimento até agora tem sido limitado pela falta de especialistas [Tamilselvam et al. 2019]. Com isso, instituições de ensino estão implementando em seus currículos, matérias com foco em IA, esperando que seus alunos consigam determinar quando é apropriado utilizar IA para resolver problemas e com isso aplicar o que foi aprendido [ACM 2013]. Porém, na prática se observam diversas dificuldades no ensino de *ML*, como por exemplo, a existência de um grande número de modelos, inúmeras bibliotecas (*Keras*, *Tensorflow*, *Pytorch, Modelzoos* e *Caffe*) e conhecimentos teóricos sobre *DL* [Tamilselvam et al. 2019]. Assim, no primeiro contato, a aprendizagem sobre *ML* torna-se difícil pelo vasto número de *frameworks* e ferramentas que o programador iniciante precisa aprender a usar para então focar no aprendizado do *ML*. Para facilitar e aumentar a velocidade da aprendizagem de conceitos básicos de *ML* poderão ser utilizados ambientes visuais, os quais estão se tornando cada vez mais comuns na programação introdutória [Weintrop et al. 2017]. Portanto, para popularizar o ML, é desejável reduzir o esforço cognitivo para que o usuário possa se concentrar na lógica para resolver o problema em questão [Knuth e Pardo 1980]. Para esse fim, foram introduzidas linguagens visuais que permitem que os usuários criem programas simplesmente arrastando e soltando um elemento visual em uma tela e, posteriormente, conectando esse elemento a outros elementos, em vez de especificá-los textualmente [Gresse von Wangenheim et al. 2021]. Tais representações visuais podem assumir diversas formas, incluindo linguagens baseadas em blocos ou baseadas em fluxo [Gresse von Wangenheim et al. 2021]. As linguagens visuais podem melhorar a capacidade de aprendizado de iniciantes, ajudando-os a evitar erros,

favorecer o reconhecimento sobre a lembrança e fornecer conjuntos de instruções limitadas específicas de domínio, reduzindo a carga cognitiva [Gresse von Wangenheim et al. 2021]. Essas vantagens levaram ao aumento da adoção em contextos de programação introdutória em diferentes estágios educacionais [Gresse von Wangenheim et al. 2021]. As ferramentas visuais também devem ajudar a preparar os alunos para, posteriormente, aprenderem linguagens baseadas em texto em ambientes mais convencionais como Jupyter Notebooks usando Python [Armoni et al. 2015][Brown et al. 2016]. Porém, mesmo surgindo cada vez mais ferramentas visuais para o desenvolvimento de modelos de ML, ainda observa-se a falta de uma ferramenta visual para ensinar o desenvolvimento de modelos de detecção de objetos, dentro do contexto de um ambiente convencional como o Jupyter Notebook (https://jupyter.org/). Então, há a possibilidade de tornar-se uma solução para facilitar a aprendizagem inicial de conceitos de *Machine Learning* também na graduação.

2. Trabalhos Relacionados

Analisando os resultados que abordavam o ensino superior foram identificadas 3 ferramentas utilizadas para ensinar *ML* neste contexto acadêmico. Provavelmente por que no ensino superior já é mais frequente a utilização de linguagens de programação textual, já que comparado com outros contextos há uma quantidade superior de exemplos de ambientes visuais e aplicações para a educação básica [Gresse von Wangenheim et al. 2021]. Assume-se que no ensino superior estes ambientes visuais são utilizados mais em momentos iniciais para ensinar a parte conceitual e no decorrer da progressão da aprendizagem dos estudantes são substituídos por ambientes textuais [Carney et al. 2020].

As ferramentas possibilitam a criação de diferentes modelos personalizados de *ML* focando principalmente em algoritmos de aprendizagem supervisionada, como somente o Milo oferecendo também suporte para a aprendizagem não supervisionada. Oferecem suporte principalmente para classificação e *clustering* de diversos tipos de dados, incluindo imagens, som, texto etc.

Os ambientes em blocos se mostraram mais eficientes para o desenvolvimento do processo de *ML* em geral, em relação aos ambientes em forma de código e tabular [Tamilselvam et al. 2019]. Com o resultado os estudantes apresentaram uma maior capacidade para o desenvolvimento de modelos mais sofisticados o suficiente para soluções mais completas [Tamilselvam et al. 2019].

Todas as ferramentas estão disponíveis online, pelo navegador (eCraft2learn, Google Teachable Machine), como aplicação *desktop* ou no navegador pela instalação do projeto no caso do Milo. Isto facilita o manuseio das ferramentas visuais, entretanto a conexão contínua com a internet pode ser um problema em alguns cenários. Em geral, as ferramentas são acompanhadas por um conjunto de tutoriais com a intenção de apoiar o estudante nos pontos importantes sobre *ML*. Porém nenhuma destas ferramentas compõem os 5 estágios para a criação de modelos de detecção de objetos como: análise de requisitos, pré-processamento de dados, transferência de aprendizado, avaliação de desempenho, predição, e exportação do modelo.

Observa-se que além dos ambientes encontrados também existem outras ferramentas visuais para desenvolvimento de modelos de *ML* como Orange (www.orangedatamining.com),

Microsoft Lobe (www.lobe.ai) e Amazon Rekognition (www.aws.amazon.com/pt/rekognition), etc. voltadas para usuários profissionais que de forma mais completa fornecem funcionalidades para diversas tarefas. Porém, como não foram encontrados relatos da aplicação destas ferramentas no ensino superior, essas não foram incluídas nesta revisão. Mesmo assim, observa-se o potencial destas ferramentas para também suportar o ensino neste nível educacional.

3. ODIN

Uma camada de interface visual para o ensino de machine learning voltada a detecção de objetos no ensino superior.

3.1. Análise de contexto

O principal objetivo da unidade instrucional é ensinar *ML* no seguinte contexto: alunos iniciantes do ensino superior principalmente no nível de graduação em cursos de computação. Assume-se que os alunos tenham um conhecimento básico de programação com linguagens textuais, como Python.

Porém, tipicamente os alunos não possuem nenhum conhecimento de *IA* nem de tarefas básicas de *ML*, como detecção de objetos. Assim, necessitam inicialmente passar por uma etapa de aprendizagem mais conceitual de conceitos de *IA*/*ML* para um processo de aprendizagem adequado.

Em relação à infraestrutura, segundo dados do diagnóstico institucional da [UFSC 2020] 93,18% dos alunos têm acesso a um computador de mesa ou notebook com acesso a internet. Porém, parte-se da premissa, que seus dispositivos podem não ser adequados para execução/treinamento das redes de DL. Sendo assim é importante que a execução não seja no computador do usuário, mas por exemplo por um servidor online gratuito, como Google Colab, o qual suportaria as atividades de desenvolvimento de *ML* pela alta demanda de processamento computacional e Google Drive para armazenamento de imagens.

3.2. Análise de requisitos

Considerando o objetivo de facilitar o aprendizado dos conceitos de ML no início da disciplina, visa-se criar uma camada visual para possibilitar a criação de modelos de *ML* de forma prática, levando o aluno ao nível de aplicação destes conceitos. Prevê-se a criação de uma camada visual sobre um Jupyter notebook para facilitar a transição posterior para um ambiente textual. Levando em consideração que já existem primeiras soluções deste tipo de camada visual para Jupyter [Franz 2021], será criado um ambiente visual para a tarefa de detecção de objetos. Visa-se utilizar *ipywidgets* (www.ipywidgets.readthedocs.io) e adaptando a interface visual criada por Franz (2021) voltada a apoiar o desenvolvimento de modelo de *ML* para tarefa de classificação de imagens.

O suporte visual para criação de modelos de *ML* para detecção de objetos será instanciado pela criação exemplar de um modelo de *ML* para detecção de objetos de casa (cadeira, porta, sapato, mesa, etc.) em termos de localização (na frente, a direita, a esquerda) visando ser um suporte para um app para ajudar pessoas com alguma deficiência visual.

A coleta de dados não será suportada pela ferramenta, prevê-se a realização do processo de rotulação/anotação das imagens via a ferramenta do *roboflow (https://roboflow.com/)* ao final o *dataset* é exportado no formato *YOLOv5 Pytorch.*

O suporte será integrado no *Jupyter Notebooks* por ser um dos ambientes textuais tipicamente adotados no ensino superior para o desenvolvimento de modelos de *ML*. Integrando o suporte diretamente no Jupyter Notebooks permitirá uma transição direta e rápida do estudante do ambiente visual para o textual no decorrer de sua progressão, reduzindo o esforço para a aprendizagem em relação ao uso das ferramentas de *ML*.

Observa-se a possibilidade de utilizar uma infraestrutura online e gratuita pelo *Google Colab* ou outra ferramenta, para que não haja dependência de computadores com uma capacidade computacional compatível com o *ML*, e para também mitigar problemas com ambiente como sistema operacional, linguagens e bibliotecas, será montado sobre o *Jupyter Notebooks/Google Colab* para facilitar e possibilitar a massiva utilização da ferramenta.

O layer visual deve permitir:

- Obtenção de imagem do *Google drive*.
- Obtenção do conjunto de dados anotados pela ferramenta *Roboflow (https://roboflow.com/)*) armazenado no Google Drive.
- Visualização das imagens do conjunto de dados (objetos por categoria, estatísticas, como a distribuição de objetos por imagens), etc.
- Treinamento de modelos *ML* selecionado o modelo, e hiperparâmetros como por exemplo *batches*, *image size* e *epochs*.
- Visualização de medidas utilizadas para analisar o desempenho referente a tarefa de detecção de objetos como, por exemplo, mean *Average Precision, precision, recall, entre outros.*
- Possibilidade de *upload* ou captura de imagens novos para testes da predição.
- Salvar o modelo treinado.
- Exportar o modelo usando o padrão *ONNX, TorchScript, Tensorflow Lite, pt, pb*.

Requisitos não funcionais:

- O ambiente precisa estar dentro do *Jupyter Notebook*.
- Treinamento precisa ser executável no ambiente online (*Google Colab*) com menos do que 2 horas (considerando um conjunto de dados com pelo menos 250 imagens)

3.3. Arquitetura do sistema

A arquitetura apresentada na Figura 1 consiste em um ambiente que rode sobre um *Jupyter Notebooks*, uma vez que um dos objetivos deste trabalho popularizar e democratizar o acesso a *IA*/*ML* e os *Jupyter Notebooks* são acessíveis diretamente pelo navegador do computador, independentes do sistema operacional. Como principal opção para viabilizar o uso deste ambiente tem-se o *Google Colab*, que é possível utilizar a plataforma de forma gratuita, sem a necessidade de instalação de programas e/ou bibliotecas adicionais, além de disponibilizar GPUs. Sobre o Colab será montado um ambiente de programação visual em que será possível realizar todas as tarefas relacionadas a ML foi utilizada a biblioteca *Jupyter Widgets* ipywidgets (2021) que é a principal referência em elementos visuais utilizados em *Jupyter Notebooks*. Para a tarefa de detecção de objetos é utilizado a biblioteca *PyTorch* que é

utilizada principalmente para o desenvolvimento de modelos de detecção de objetos YOLOv5, que tem total integração com o Google Colab.

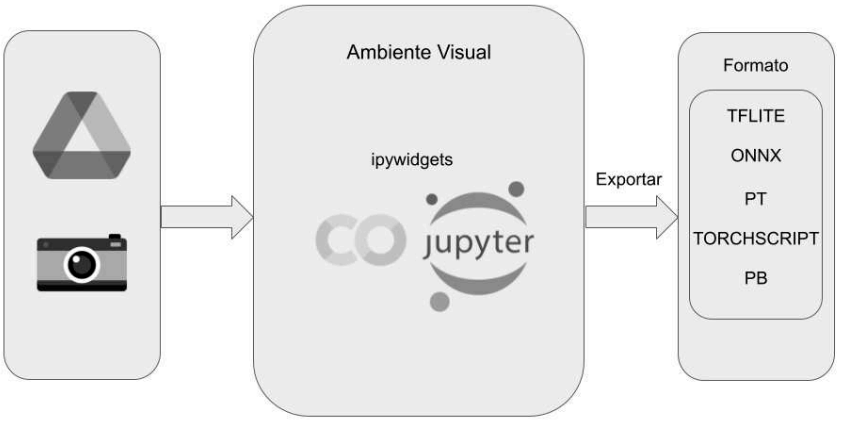


Figura 1 - Visão geral da arquitetura do sistema

Apesar de suportar elementos visuais, o Colab não foi concebido como uma interface gráfica ou aplicação de uma célula. Normalmente se utilizam diversas células, uma com cada trecho de código referente a etapa do processo de ML. Entretanto é possível encapsular todos os processos de IA/ML de acordo com os elementos visuais disponíveis e executar a aplicação de ponta a ponta em apenas uma célula, escondendo em si o código da aplicação como demonstrado na Figura 2.

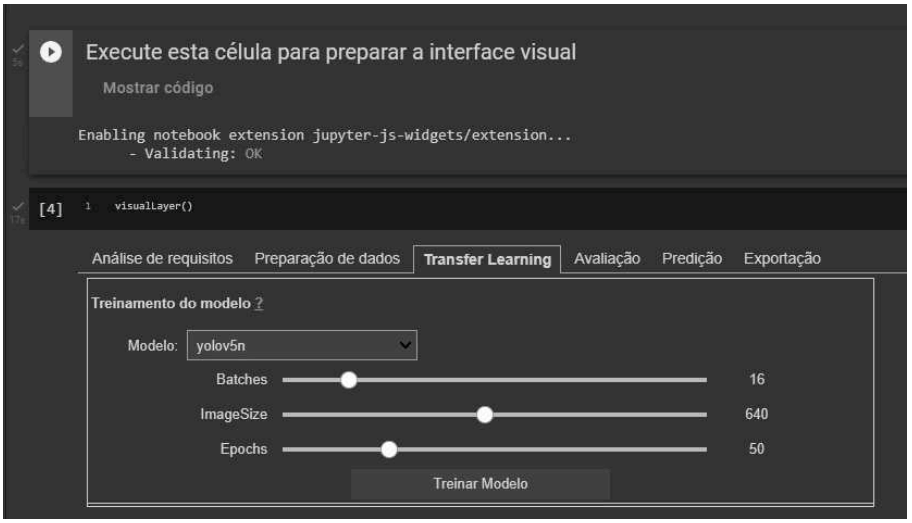


Figura 2 - Interface do ODIN

3.4. Implementação do ODIN

Nesta seção é apresentado o desenvolvimento e implantação da biblioteca. Toda a estrutura proposta foi desenvolvida utilizando Python (https://www.python.org/) por rodar nativamente no colab e ter total compatibilidade com os frameworks selecionados. De acordo com a análise de requisitos e a arquitetura definida seguindo o human-centric interactive ML process [Gresse Von Wangenheim e Von Wangenheim 2021] foram especificadas as

necessidades de entrada e saída de dados guiando o suporte desenvolvido *ODIN* conforme apresentado na Tabela 1.

Tabela 1 - Definição do suporte do ODIN

Fase do Processo	**Passo do Processo**	**Detalhes e Padrões**
Análise de Requisitos	Especificação da(s) tarefa(s) e requisitos do sistema de ML	● Tarefa ● Tipo da tarefa ● Categorias ● Fonte de dados ● Quantidade de dados ● Padronização das imagens ● Rotulação dos dados ● Desempenho
Preparação do Dados	Conjunto de dados anotado (no Google Drive)	Um conjunto de dados de imagens já anotado (com formato para YOLOV5) deve ser disponibilizado em uma pasta no Google Drive e dentro desta pasta deve haver 3 pastas e um arquivo .yaml: uma pasta para treinamento, uma para validação e outra para teste, e um arquivo de configuração dentro de cada respectiva pasta é preciso ter 2 pastas, uma para as imagens e a outra para os labels.
	Definição do conjunto de dados	Caminho do diretório no Google Drive
	Visualizar informações do conjunto de dados	Percorre o conjunto de dados total informado e mostra um histograma com a distribuição dos objetos por classe
	Visualizar conjunto de dados	Percorre o conjunto de dados de treinamento informado e mostra até 6 exemplos com suas respectivas labels
Transfer Learning	Seleção do modelo	Padrão = yolov5n ● yolov5n ● yolov5s ● yolov5m ● yolov5l ● yolov5x
	Tamanho do batch	Padrão = 16 ● min = 1 ● max = 128
	Tamanho das imagens	Padrão = 640 ● min = 64 ● max = 1280
	Quantidade de épocas	Padrão = 50 ● min = 1 ● max = 500
	Treinar modelo	Executa o treinamento do modelo escolhido a partir da quantidade de épocas, tamanho dos batches e tamanho de imagem informada.
Avaliação	Avaliação do treinamento	Exibe as principais métricas para validar o treinamento do modelo: ● train/box_loss ● train/obj_loss ● train/cls_loss ● metrics/precision ● metrics/recall ● val/box_loss ● val/obj_loss ● val/cls_loss

		• metrics/mAP_0.5 • metrics/mAP_0.5:0.95
	Curva do F1 score	Exibe o F1 Score
	Matriz de confusão	Exibe a matriz de confusão
Predição	Taxa de confiança	Padrão = 10 • min = 1 • max = 100
	Predição pelo conjunto de teste	Faz a predição pelas imagens da pasta test do Dataset
	Predição pelo *upload*	Permite o upload pelo HD de uma imagem (por vez) para realizar a predição da mesma.
Exportação	Seleção de formato	Padrão = pt • tflite • torchscript • pb • onnx • pt
	Definição do caminho	Caminho do diretório no Google Drive para exportar o modelo no formato
	Exportar	Executa o algoritmo para transformar para o formato escolhido e cópia para a pasta definida no passo anterior

3.5. Integração com o Google Drive

A integração com o Google Drive [Colab 2021] é a ponte entre a entrada e saída dos dados com o *ODIN*, permitindo tanto a entrada do conjunto de dados para o treinamento do modelo quanto a exportação do modelo treinado.

A autenticação é feita com a API da Google e é possível logar com a conta que desejar (não necessariamente a mesma do Colab), esta conta se conectará ao drive apenas nas pastas informadas pelo próprio usuário durante a execução.

Para integrar o conjunto de dados deve conter 3 pastas, uma para treinamento, uma validação e outra para teste e um arquivo *.yaml* de configuração.

3.6. Biblioteca Ultralytics/yolov5

Utilizada uma série de arquivos da biblioteca do Ultralytics/yolov5 (github.com/ultralytics/yolov5) para realizar o processo completo de ML. São utilizados os seguintes objetos da biblioteca:

- train.py é o arquivo utilizado para treinar o modelo, as informações utilizadas para execução do treino são passadas por parâmetro como batches, imageSize, epochs entre outros;
- detect.py é o arquivo utilizado para detectar objetos em uma imagem, as informações utilizadas na execução são passadas por parâmetro, como taxa de confiança, modelo, entre outros.
- val.py é o arquivo utilizado para avaliar o modelo treinado retornando o mAP de geral e para cada classe do conjunto de dados.

3.7. Processo de ML

O fluxo do processo de ML no *Visual Layer* foi desenvolvido seguindo o processo de desenvolvimento de ML centrado no ser humano conforme apresentado na Figura 3 [Gresse von Wangenheim e von Wangenheim 2021], e também nas necessidades da biblioteca do *Ultralytics/yolov5* apresentadas na seção 2.1. Para desenvolver essa camada intermediária foram criadas algumas funções para garantir os objetos necessários além da execução das etapas pré-definidas.

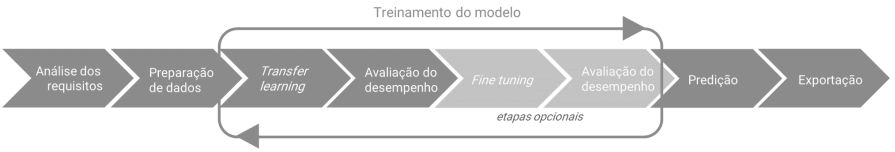


Figura 3 - Etapas do processo de ML centrado no ser humano

create_config_files(): Responsável pela atualização do arquivo *data.yaml,* alterando o caminho definido no arquivo para as pastas *train* e *valid* do dataset definido pelo usuário;

charge_yolov5(): Responsável pela importação do projeto do *YOLOv5* e instalação de seus pacotes requeridos;

drive_connect(): Responsável pela abertura de conexão com o Google Drive e definição e gravação do caminho raiz na variável *root_dir*.

render_odin(): Responsável pela renderização de todos os componentes e funções intermediárias como charge_yolov5() e drive_connect();

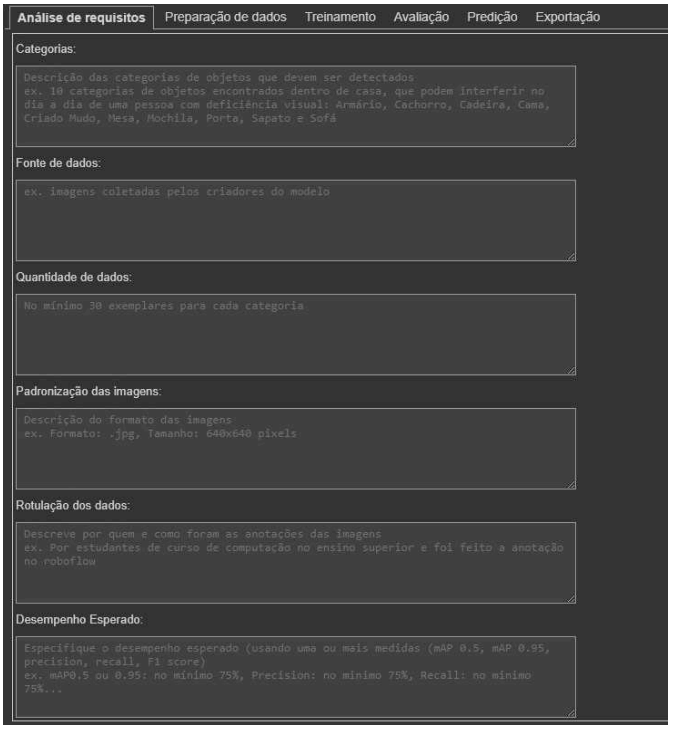
3.8. Interface visual

A interface representa visualmente todo o processo de ML/IA na ferramenta, integrando todos os elementos. Para criar um ambiente sem código, tudo foi encapsulado na função *render_odin()*.

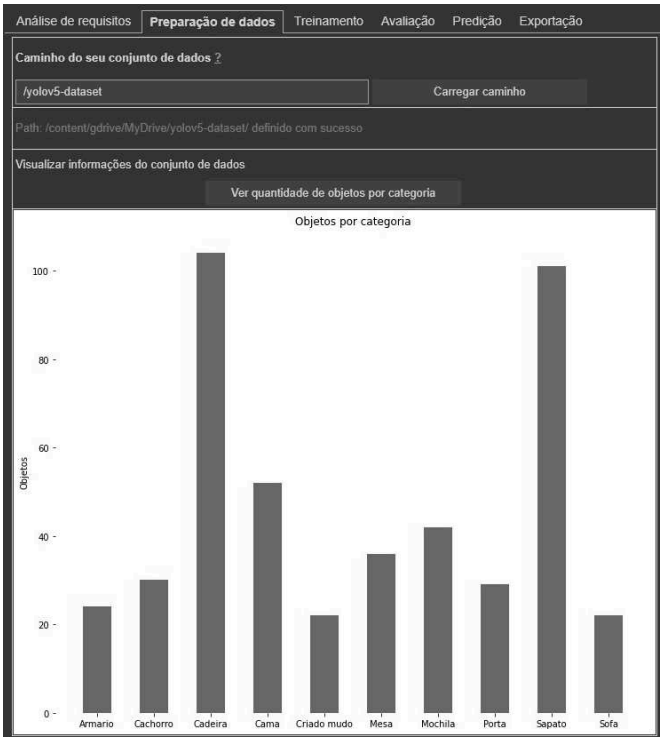
A base interface visual da ferramenta foi criada com a biblioteca do Google Colab widgets. Foram utilizados tabs para criação das abas e grids para conter os visuais e saídas da ferramenta. Para os demais elementos (textos, entradas e botões) foi utilizado a biblioteca ipywidgets com seus diversos *widgets*, *styles*, *layouts* e interações. A Tabela 2 mostra o resultado da interface visual desenvolvida (https://codigos.ufsc.br/gqs/odin).

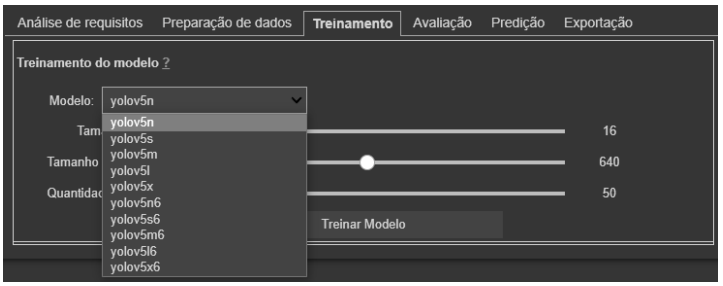
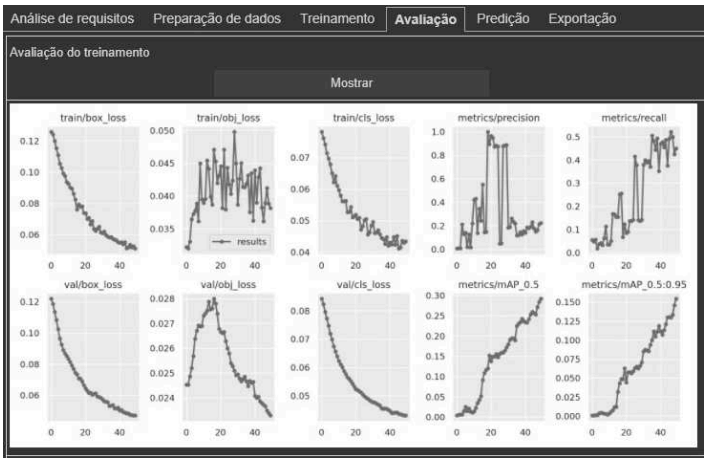
Tabela 2 - Interface visual do ODIN

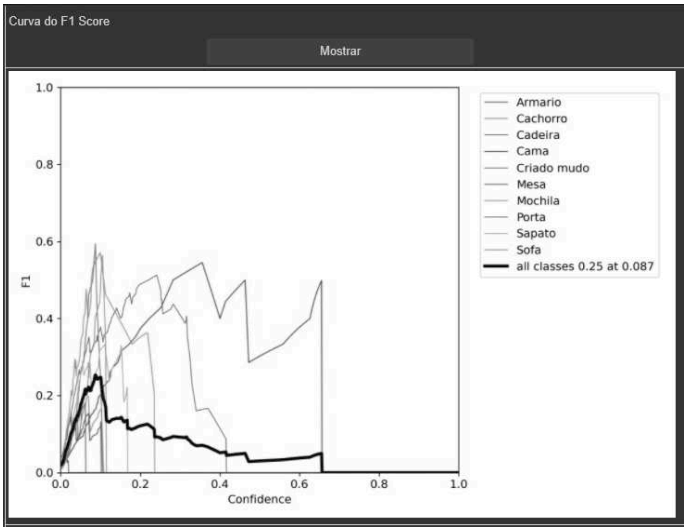
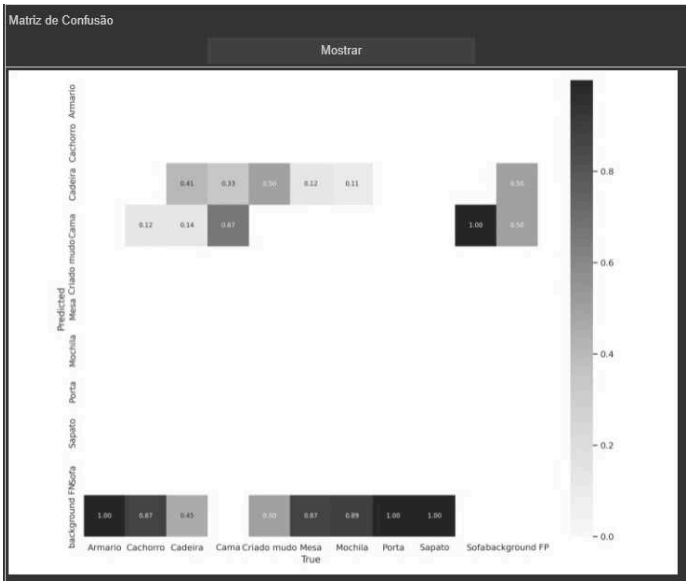
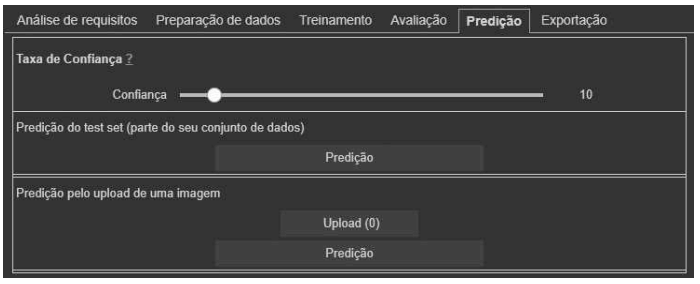
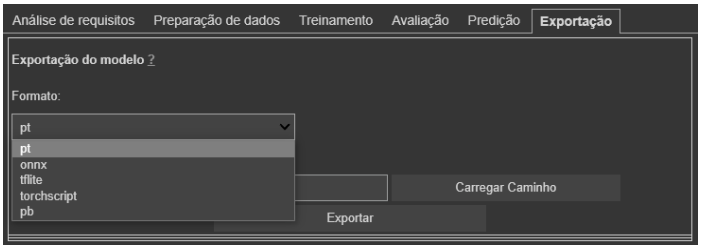
Fase do Processo	**Passo do Processo**	**Interface visual**

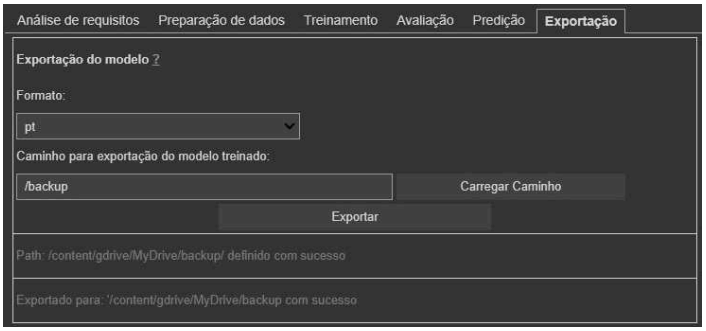
<table>
<tr>
<td>Análise de Requisitos</td>
<td>Especificação da(s) tarefa(s) e requisitos do sistema de ML</td>
<td> 
</td>
</tr>
<tr>
<td>Preparação dos Dados</td>
<td>Conjunto de dados (no Google Drive)

Definição do conjunto de dados

Visualizar informações do conjunto de dados</td>
<td> 
</td>
</tr>
</table>

	Visualizar conjunto de dados	
Transfer Learning	Seleção do Modelo	
	Tamanho do batch Tamanho das imagens Quantidade de épocas Treinar modelo	
Avaliação do desempenho	Avaliação do treinamento	

	Curva do F1 Score	
	Matriz de confusão	
Predição	Taxa de confiança	
	Predição pelo conjunto de teste	
	Predição pelo *upload*	
Exportação	Seleção do formato	

	Definição do caminho	
	Exportar	

4. Conclusão

Neste trabalho de conclusão de curso, foram apresentados os conceitos de *machine learning* e buscadas ferramentas visuais com o propósito de instruir estudantes do ensino superior a iniciarem seu desenvolvimento na área de *machine learning* como no caso da ferramenta do *Google Teachable Machine*.

Com essa base de conhecimento foi implementada uma camada visual a qual suporta o processo de desenvolvimento de sistemas de *Machine Learning* no contexto do ensino superior, a ferramenta ODIN é capaz de criar modelos sem a necessidade de código textual por meio de 6 fases para o desenvolvimento: análise de requisitos, preparação dos dados, treinamento, avaliação, predição e exportação. O ODIN é uma ferramenta prática que abstrai as dificuldades da configuração do projeto e organiza o processo da criação até a exportação do modelo. Para trabalhos futuros planeja-se a adição do passo de *fine tuning* (ajuste fino), a inclusão de mais hiperparâmetros para customizar o treinamento, *augmentation* para equilibrar o dataset, realização de uma avaliação do desempenho da ferramenta e a integração a um curso no ensino superior.

REFERÊNCIAS

ACM, Computer Science Curricula, 2013. Disponível em: <https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf> Acesso em Maio de 2020.

S. Amershi, et al. 2019. Software Engineering for Machine Learning: A Case Study. In Proc. of the 41st International Conference on Software Engineering: Software Engineering in Practice, IEEE Press, 291–300.

Amazon. 2019. Amazon Machine Learning, AWS Documentation. https://docs.aws.amazon.com/machine-learning/latest/dg/building-machine-learning.html.

M. Armoni, O. Meerbaum-Salant, e M. Ben-Ari. 2015. From Scratch to “Real” Programming.

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

V. Bushaev ; How do we ‘train’ neural networks?, 2017. Disponível em:<https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94> . Acesso em Agosto de 2020.

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

M. Carney, B. Webster, I. Alvarado, K. Phillips, N. Howell, J. Griffith, J. Jongejan, A. Pitaru, and A. Chen. 2020. Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification. In Proc. of Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems. ACM, New York, NY, USA, 1–8.

I. Costa , E. Silva Jr , A. Rodrigues , L. Angeloni e E. Dias. 2020. Avaliação do Processo para Embarcar uma Rede Neural Baseada em YOLO Utilizando um Acelerador de Hardware Dedicado.

M. Copeland. What 's the Difference Between Artificial Intelligence, Machine Learning and Deep Learning? Disponivel em: <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/> Acesso em Maio de 2020.

P. Dorotea, Nuno & Sampaio, Fábio & Pedro, Ana. (2019). A Cross-analysis of Block-based and Visual Programming Apps with Computer Science Student-Teachers. Education Sciences, 9, 181.

J. Dudley, P. Kristensson 2018. A Review of User Interface Design for Interactive Machine Learning.ACM Trans. Interact. Intell. Syst,8(8), 2018.

Fiebrink, Rebecca & Gillies, Marco. (2018). Introduction to the Special Issue on Human-Centered Machine Learning. ACM Transactions on Interactive Intelligent Systems. 8. 1-7.

C. Gresse von Wangenheim, J. Hauck, F. Pacheco, et al. Visual tools for teaching machine learning in K-12: A ten-year systematic mapping. Educ Inf Technol 26, 5733–5778 2021.

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

M. Gillies, et al. (2016) Human-Centered Machine Learning. Proc. of the Conference on Human Factors in Computing Systems, ACM, San Jose, CA, USA, 3558-3565, https://hal.inria.fr/hal-01437057/document.

L. Goasduff. This Gartner Hype Cycle highlights how AI is reaching organizations in many different ways, 2019.Disponivel em <https://www.gartner.com/smarterwithgartner/top-trends-on-the-gartner-hype-cycle-for-artificial-intelligence-2019/> Acesso em Maio de 2020

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

W. Hechun and Z. Xiaohong. Survey of Deep Learning Based Object Detection. In Proceedings of the 2nd International Conference on Big Data Technologies. ACM, New York, NY, USA, 2019., 149–153.

A. Hoover, A. Spryszynski, and M. Halper. 2019. Deep Learning in the IT Curriculum. In Proc. of the 20th Annual SIG Conference on Information Technology Education . ACM, New York, NY, USA, 49–54.

D. Hils, Visual languages and computing survey: Data flow visual programming languages. 1992. J. Vis. Lang. Comput., vol. 3, no. 1, pp. 69–101, Mar.

C. Hmelo e M. Guzdial. Of black and glass boxes: scaffolding for doing and learning. In Proc. of the international Conference on Learning sciences. International Society of the Learning Sciences, 1996 , 128–134.

L. Jiao et al. A Survey of Deep Learning-Based Object Detection. IEEE Access, 7, 128837-128868, 2019.

J. Johnson. 2019. Teaching neural networks in the deep learning era. J. Comput. Sci. Coll., 34(6) 2019, 16–25.

J. Johnson. 2020. Benefits and Pitfalls of Jupyter Notebooks in the Classroom. In Proceedings of the 21st Annual Conference on Information Technology Education. Association for Computing Machinery, New York, NY, USA, 32–37.

W. Johnston, J. Hanna, R. Millar Advances in Dataflow Programming Languages. 2004. ACM Comput. Surv., vol. 36, no. 1, pp. 1–34, Mar.

T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, et al. 2016. Jupyter

notebooks – a publishing format for reproducible computational workflows. (Loizides F& Schmidt B, Eds.).

J. Zhang, P. Qu, C. Sun et al., “DWCA-YOLOv5: An improve single shot detector for safety helmet detection,” Journal of Sensors, 2021

D. Knuth, L. Pardo, 1980. The early development of programming languages. In A history of computing in the twentieth century, 197–273.

K. Kahn, N. Winters, Child-Friendly Programming Interfaces to AI Cloud Services. 2017. In: Lavoué É. et al. (eds) Data Driven Approaches in Digital Education. Lecture Notes in Computer Science, vol 10474. Springer, Cham.

P. Karkare; Convolutional Neural Networks – Simplified, 2019. Acesso em: Agosto de 2021

B. Kenji, 2019; Machine Learning para Leigos. Disponível em: Machine Learning para Leigos | Venturus. Acesso em: Julho de 2021.

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

Y. Lecun, Y. Bengio, G. Hinton. Deep Learning, 2015. Nature, 521, 436-44.

MIT App Inventor. About Us, 2019. Disponível em: About Us (mit.edu) Acesso em: Julho de 2021.

R. Meulen e C. Pettey, Gartner Says by 2020, Artificial Intelligence Will Create More Jobs Than It Eliminates.Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2017-12-13-gartner-says-by-2020-artificial-intelligence-will-create-more-jobs-than-it-eliminates > Acesso em Maio de 2020

K. Mathewson, (2019). A Human-Centered Approach to Interactive Machine Learning. arXiv:1905.06289v1 [cs.HC].

M. Nelson e A. Hoover. 2020. Notes on Using Google Colaboratory in AI Education. In Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20). Association for Computing Machinery, New York, NY, USA, 533–534.

G. Nodalo, J. Santiago, J. Valenzuela, and J. Aiko Deja. 2019. On Building Design Guidelines for An Interactive Machine Learning Sandbox Application. In Proc. of the 5th International ACM In-Cooperation HCI and UX Conference,ACM, New York, NY, USA, 70–77.

J. Perkel. 2018. Why Jupyter is data scientists’ computational notebook of choice. Nature 2018.

J. Piazentin Ono, J. Freire and C. Silva, "Interactive Data Visualization in Jupyter Notebooks," in Computing in Science & Engineering, vol. 23, no. 2, pp. 99-106, 1 Mar 2021.

E. Pasternak, R. Fenichel and A. N. Marshall.Tips for creating a block language with blockly," 2017 IEEE Blocks and Beyond Workshop, Raleigh, NC, 2017, 21-24.

Pathmind, A Beginner’s Guide to Neural Networks and Deep Learning. Disponível em:<https://pathmind.com/wiki/neural-network>. Acesso em Julho de 2020a.

Pathmind, A Beginner's Guide to LSTMs and Recurrent Neural Networks. Disponível em:<https://pathmind.com/wiki/lstm>. Acesso em Julho de 2020.

Pathmind, A Beginner’s Guide to Convolutional Neural Networks (CNNs). Disponível em:<https://pathmind.com/wiki/convolutional-network>. Acesso em Julho de 2020c.

Pant, A. Workflow of a Machine Learning Project, 2019. Disponível em: <https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94?gi=ef53a0d0e22>. Acesso em Agosto de 2020

R. Padilla, S. Netto e E. da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms," 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), 2020.

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

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

Robinson, 2018. Sketch2code: Generating a website from a paper mockup. Dissertation, University of Bristol, UK.

G. Ramos, C. Meek, P. Simard, J. Suh, S. Ghorashi. (2020) Interactive machine teaching: a human-centered approach to building machine-learned models, Human–Computer Interaction, 35(5-6).https://www.microsoft.com/en-us/research/uploads/prod/2020/05/Interactive_Machine_Teaching__Free_access_.pdf

G. Ramos, C. Meek, P. Simard, J. Suh, S. Ghorashi. (2020) Interactive machine teaching: a human-centered approach to building machine-learned models, Human–Computer Interaction, 35(5-6).https://www.microsoft.com/en-us/research/uploads/prod/2020/05/Interactive_Machine_Teaching__Free_access_.pdf

S. Russell e P. Norvig Artificial Intelligence A Modern Approach: Englewood Cliffs, New Jersey: Prentice-Hall, 1995.

R. Sinnott, D. Yang, X. Ding, and Z. Ye. 2020. Poisonous Spider Recognition through Deep Learning. In Proc. of the Australasian Computer Science Week Multiconference. Association for Computing Machinery, New York, NY, USA, Article 14, 1–7

S. Tamilselvam, N. Panwar, S. Khare, R. Aralikatte, A. Sankaran, S. Mani. A visual programming paradigm for abstract deep learning model development. In Proc. of the 10th Indian Conference on Human-Computer Interaction. ACM, New York, NY, USA, 2019. Article 16, 1–11.

UFSC, Diagnóstico Institucional UFSC Junho 2020. Disponível em: 15_06_20_DIAGNOSTICO_INSTITUCIONAL_1.pdf (ufsc.br). Acesso em Junho de 2021.

A. Wangenheim. Detecção de objetos em imagens. 2018 Disponível em: Deep Learning::Detecção de Objetos em Imagens (ufsc.br). Acesso em 2021 de Agosto.

Wissal Farsal, Samir Anter, and Mohammed Ramdani. 2018. Deep Learning: An Overview. In Proceedings of the 12th International Conference on Intelligent Systems: Theories and Applications (SITA'18). ACM, New York, NY, USA, Article 38, 1–6.

D. Weintrop, W. Uri.(2017). How block-based languages support novices: A Framework for categorizing block-based affordances. Journal of Visual Languages and Sentient Systems, 3. 92-100.

D. Weintrop, H. Killen, T. Munzar, and B. Franke. 2019. Block-based Comprehension: Exploring and Explaining Student Outcomes from a Read-only Block-based Exam. In Proc. of the 50th ACM Technical Symposium on Computer Science Education. Association for Computing Machinery, New York, NY, USA, 1218–1224.

Y. Watanabe, et al. (2019). Preliminary Systematic Literature Review of Machine Learning System Development Process. arXiv:1910.05528 [cs.LG].

D. Weintrop. Block-based Programming in Computer Science Education". Communications of the ACM, 62(8), 2019, 22-25.

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