UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
DEE - DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO BACHAREL EM ENGENHARIA ELÉTRICA

PEDRO IVO ARAGÃO GUIMARÃES

TÉCNICAS DE APRENDIZAGEM PROFUNDA PARA ESTACIONAMENTOS INTELIGENTES

TRABALHO DE CONCLUSÃO DE CURSO

CAMPINA GRANDE
2021

PEDRO IVO ARAGÃO GUIMARÃES

TÉCNICAS DE APRENDIZAGEM PROFUNDA PARA ESTACIONAMENTOS INTELIGENTES

Trabalho de Conclusão de Curso apresentado ao Curso bacharel em Engenharia Elétrica da Universidade Federal de Campina Grande, como requisito parcial para a obtenção do título de Bacharelado.

Orientador: Dr. Rafael Bezerra Correia Lima
Universidade Federal de Campina Grande

CAMPINA GRANDE
2021

Este trabalho tem uma dedicatória especial ao meu grande amigo Rodolpho Luiz Barros de Medeiros que descansou nos braços de Deus durante a realização deste trabalho, mas sei que de onde estiver, estará feliz pela minha conquista. E aos meus pais Sérgio Guimarães e Margareth Ribeiro Aragão e a minha irmã Milena Aragão que sempre me apoiaram e não mediram esforços para eu chegar até aqui. Amo todos vocês!

AGRADECIMENTOS

Primeiramente à Deus, sem ele nada seria possível. Agradecimento especial à todos os professores do DEE pelo compartilhamento do conhecimento científico, moral e ético. A Universidade Federal de Campina Grande responsável pela infraestrutura e qualidade de todos os profissionais que contribuíram para a minha longa jornada. Ao SENAI por abrir as portas técnicas que despertou o interesse pela área do conhecimento. Aos meus colegadas de curso, que coletivamente compartilharam conhecimentos e tornou cada etapa mais fácil. Minha namorada Me. Rayanne Izabel Maciel que me apoiou em todos os momentos, nas alegrias e nas tristeza durante esta caminhada e devo grande parte dessa conquista a ela. Meus colegas de trabalho da Vsoft: Rogério lucas Marinho, Rafael Marques, Sandoval Neto, Dra. Anna Medeiros, Dr. João Janduy pelo compartilhamento diário de conceitos relevantes para o tema, que tornou tudo mais fácil, em especial ao Me. Guilherme Monteiro Gadelha pelas sugestões e correções deste trabalho, por fim, não menos importante, o professor Dr. Rafael Bezerra que me despertou o interesse pela pesquisa e esteve sempre solícito a contribuir de forma grandiosa com a realização deste trabalho

Não creio que haja uma emoção mais intensa para um inventor do que ver suas criações funcionando. Essas emoções fazem você esquecer de comer, de dormir, de tudo. (TESLA, Nikola).

RESUMO

Estacionar carros nas grandes cidades tem se tornado um enorme desafio, visto que o número crescente de automóveis privados adquiridos nas últimas décadas contribui para a falta de mobilidade urbana. Frequentemente os motoristas desperdiçam muito mais tempo que o necessário para estacionar os carros, contribuindo para o estresse no trânsito. A inteligência artificial vem buscando métodos para solucionar este problema, isto só se tornou possível nos dias atuais pela disponibilidade de grandes quantidades de dados e a melhoria na capacidade de processamento dos computadores. Uma técnica de sistema de visão com forte tendência nos últimos anos é o aprendizado profundo com algoritmos de redes convolucionais. Este trabalho visa avaliar técnicas de aprendizagem profunda para a construção e avaliação de um modelo para estacionamentos inteligentes, utilizando um conjunto de dados da UFPR, em três cenários distintos. O modelo realiza a predição em tempo real e propõe um melhor custo-beneficio do que alternativas com internet das coisas, dispensando modificação no ambiente e de fácil manutenção.

Palavras-chave: Estacionamento Inteligente. Aprendizado profundo. Sistema de Visão computacional.

ABSTRACT

Parking in big cities has become a huge challenge, as the growing number of private cars purchased in recent decades contributes to the lack of urban mobility. Drivers often waste far more time than necessary to park their cars, contributing to traffic stress. Artificial intelligence has been looking for methods to solve this problem, this has only become possible nowadays due to the availability of large data resources and an improvement in the processing capacity of computers. A system of vision technique with a strong trend in recent years is deep learning with convolutional network algorithms. This work aims to evaluate deep learning techniques for the construction and evaluation of a model for intelligent parking, using a dataset from UFPR, in three distinct scenarios. The model performs the prediction in real-time and proposes a better cost-benefit ratio than alternatives with the Internet of Things, eliminating changes in the environment and being easy to maintain.

Keywords: Smart parking. Deep Learning. Computer Vision.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IA	Inteligência Artificial.
ML	Aprendizado de máquinas (*Machine Learning*).
RNA	Redes Neurais Artificiais.
CNN	Redes Neurais Convolucionais (*Convolutional Neural Network*).
MLP	Perceptron de multi camadas (*MultiLayer Perceptron*).
LTU	Unidade linear de limiar (*A Linear Threshold Unit*).
XOR	Ou exclusivo (*Exclusive Or*).
MSE	Erro quadrático médio (*Mean Squared Error*).
UFPR	Universidade Federal do Paraná.
PUCPR	Pontifícia Unidade Católica do Paraná.
ROI	Região de interesse (*Region of Interest*).

LISTA DE SÍMBOLOS

∇	Vetor Gradiente
$\sum$	Somatório

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

A melhoria da conectividade e das redes de computadores tem proporcionado o desenvolvimento de cidades inteligentes (*smart cities*) que vem conquistando mais espaços no cenário atual. De acordo com a União Europeia, cidades inteligentes são sistemas de pessoas interagindo e usando energia, materiais, serviços e financiamento para catalisar o desenvolvimento econômico e a melhoria da qualidade de vida.

Esses fluxos de interação são considerados inteligentes por fazerem uso estratégico de infraestrutura e serviços de informação e comunicação, com planejamento e gestão urbana para dar resposta às necessidades sociais e econômicas da sociedade (FGV, 2019). Portanto, utilizam a tecnologia de modo estratégico para melhorar a infraestrutura, otimizar a mobilidade urbana, criar soluções sustentáveis e outras melhorias necessárias para a qualidade de vida da sociedade (PANHAN, 2020). A subcategoria que será o tema abordado neste projeto são os *smart parking*, sistemas inteligentes para auxiliar motoristas na localização de vagas de estacionamento.

1.1 DEFINIÇÃO DO PROBLEMA.

Estacionar o carro nas grandes cidades tem se tornado um enorme desafio, visto que o carro privado tornou-se um modo de transporte bastante atrativo. Entre os anos de 2008 e 2018, houve um número crescente de automóveis no país, que passou de 37,1 milhões para 65,7 milhões, desse crescimento, destaca-se que 40% dos veículos estão localizados em um número reduzido de regiões metropolitanas. Isso implica dizer que as cidades que enfrentam com a falta de mobilidade urbana e consequentes problemas de trânsito continuam recebendo boa parte da carga de novos automóveis (AZEVEDO; RIBEIRO, 2019).

Neste contexto, o desenvolvimento de sistemas inteligentes para gerenciar as vagas de estacionamentos torna-se fundamental para garantir uma melhoria na qualidade de vida das cidades. Um estudo relatou que 30% do congestionamento do tráfego é causado por buscas de locais para estacionar, que duram cerca de 7 a 8 minutos por tentativa (ARNOTT; INCI, 2006). Tal fato leva a um maior consumo de combustível e consequente aumento da poluição do ar, da emissão de gás carbônico, além de gerar congestionamento do tráfego (SIECK; CALPIN; ALMALAG, 2020). Desse modo, os sistemas inteligentes podem ser úteis por meio do fornecimento de informações em tempo real aos motoristas sobre a disponibilidade e a localização das vagas de um estacionamento, reduzindo o tempo gasto para encontrar as vagas disponíveis no momento e todos os transtornos causados em consequência dessa busca.

1.1.1 PREMISSAS E HIPÓTESES.

O problema descrito pode ser minimizado e automatizado com a implementação de um sistema de visão computacional, que é um campo de pesquisa dentro de inteligência artificial (IA), onde a partir do tratamento de imagens digitais os computadores processam e fornecem informações relevantes para o sistema. A visão computacional permite que eles vejam, observem e compreendam (DEMUSH, 2019).

Esta solução torna o projeto economicamente viável sendo necessária uma mínima mudança no espaço e infraestrutura. Poucos equipamentos, como câmeras conectadas à internet, muitas vezes reaproveitadas de outras funções, e um servidor para funcionamento do modelo computacional. Há também um beneficio diretamente proporcional com a manutenção do sistema.

Desse modo, espera-se um ganho relativo do sistema, comparado a modelos tradicionais de estacionamentos inteligentes com sensores e microcontroladores.

1.1.2 OBJETIVO GERAL.

Oferecer um modelo classificador para vagas de estacionamentos, com técnicas de aprendizagem profunda, para auxiliar a mobilidade urbana das cidades inteligentes.

1.1.3 OBJETIVOS ESPECÍFICOS.

- Ser facilmente integrado com os sistemas de câmeras convencionais.
- Treinar um modelo em um banco de dados específico que possa ser generalizado e reaproveitado para outros ambientes.
- Avaliar a eficiência de estacionamento inteligentes unicamente com sistemas de visão.
- Comparar resultados com outras literaturas de *smart parking* com visão computacional.

1.2 ORGANIZAÇÃO DO TRABALHO.

Este trabalho está dividido em três etapas, revisão da literatura, metodologia e análise e discussão dos resultados:

1. Revisão da literatura - Será apresentado um breve conceito sobre redes neurais artificiais: história e arquitetura do perceptron de uma e mais camadas, conceito de gradiente descendente, algoritmos do *forward* e *backpropagation*, funções de ativações e erro e redes convolucionais.
2. Metodologia[1] - É composta pela aquisição do banco de dados, pré-tratamento das imagens, rotulação, hiper parâmetros e treinamento do classificador supervisionado.
3. Resultados e Discussão - Resultados e métricas alcançados durante este trabalho para a detecção de vagas livres com visão computacional, projeção de melhorias.

[1] Todo o material de código-fonte deste projeto está disponível neste link no github em um repositório público com o passo a passo para a reprodução.

2 FUNDAMENTOS TEÓRICOS.

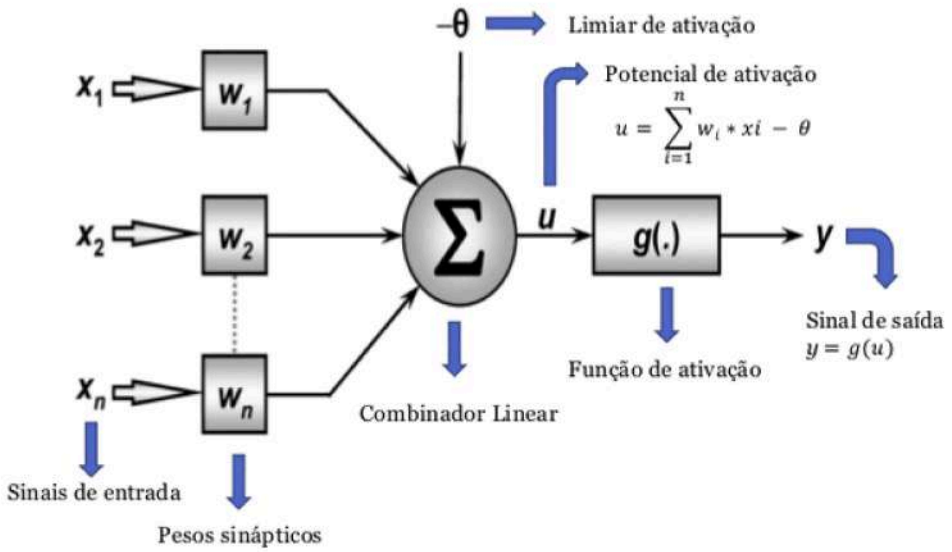
2.1 REDES NEURAIS ARTIFICIAIS (RNA).

As Redes neurais artificiais (RNA) constituem um tipo específico de algoritmo de inteligencia computacional. É um braço da IA, mais especificamente está contido em aprendizagem de máquinas ou *machine learning* (ML). ML é a ciência e arte onde os computadores são preparados para aprender a partir de dados. Uma definição clássica da literatura para engenheiros: "Diz-se que um programa de computador aprende pela experiência E em relação a algum tipo de tarefa T e alguma medida de desempenho P se o seu desempenho em T, conforme medido por P, melhora com a experiência E"(MITCHELL, 1997).

As RNAs são inspiradas no comportamento do cérebro humano (neurônios e sinapses), a semelhança vem de utilizar elementos simples (neurônios) para compor um sistema mais complexo, por meio das ligações das sinapses. No caso das redes artificiais, a partir da composição de funções lineares ligadas em rede, modelar sistemas complexos. Segundo Nielsen (1989), "Um sistema de computação constituído por uma série de simples elementos de processamento altamente interligados, que processam informações por sua resposta de estado dinâmico a entradas externas"

Uma RNA com uma camada e um neurônio, as entradas X_1,X_2..X_n são multiplicadas com seus respectivos pesos sinápticos e somadas, posteriormente é composta por outra função, chamada de função de ativação e o resultado do sistema é a saída Y (Figura 1). Essa arquitetura é conhecida como perceptron e será mais detalhada no próximo tópico.

Figura 1 – Exemplo de RNA de uma camada.



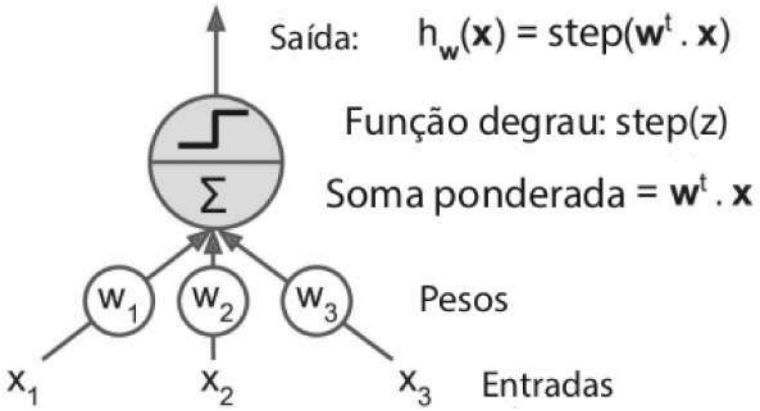
Fonte: Palmiere (2016)

2.2 PERCEPTRON

Entre os anos de 1940 até 1958, o conceito de RNA era pouco relevante e as entradas eram apenas valores binários. Aproximadamente 18 anos após o surgimento, o americano Frank Rosenblatt trouxe uma arquitetura simples que é usada como exemplo didático até os dias atuais.

A arquitetura do perceptron é composta apenas por um neurônio artificial diferente, chamado de unidade linear de *threshold(LTU)*, as entradas e saídas são números e cada conexão de entrada está associado a um peso. A LTU calcula a soma ponderadas das entradas e aplica uma função degrau a esta soma, como visto na Figura 2. O percepetron é capaz de resolver apenas problemas linearmente separáveis com classificação binária (GéRON, 2017).

Figura 2 – Perceptron.



Fonte: Géron (2017)

Algoritmo 1: Perceptron

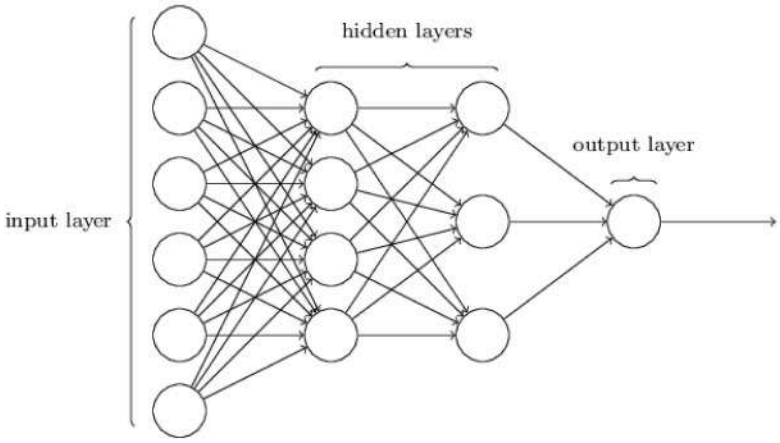
Input: X_1, X_2, X_3
Output: $h_w(X)$
sinapse1 $= X_1 * W_1$
sinapse2 $= X_2 * W_2$
sinapse3 $= X_3 * W_3$
resultado $= \sum_{i=0}^{3} sinapse1 + sinapse2 + sinapse3$
$h_w(x) = heaviside(resultado)$

O Algoritmo 1 proposto não é capaz de solucionar um problema simples como a porta XOR (Figura 5). Problemas não lineares podem ser solucionados com a adição de camadas ocultas, que deu origem ao nome *deep learning* ou aprendizagem profunda. Essa é a razão de Frank Rosenblatt ser considerado um dos pioneiros do aprendizado profundo.

2.3 PERCEPTRON DE MULTICAMADAS (MLP).

O perceptron de multi camadas (MLP) é uma arquitetura de RNA (Figura 3) com uma ou mais camadas ocultas e com um número indeterminado de neurônios. São adicionadas mais camadas e LTU's para solução de problemas não lineares.

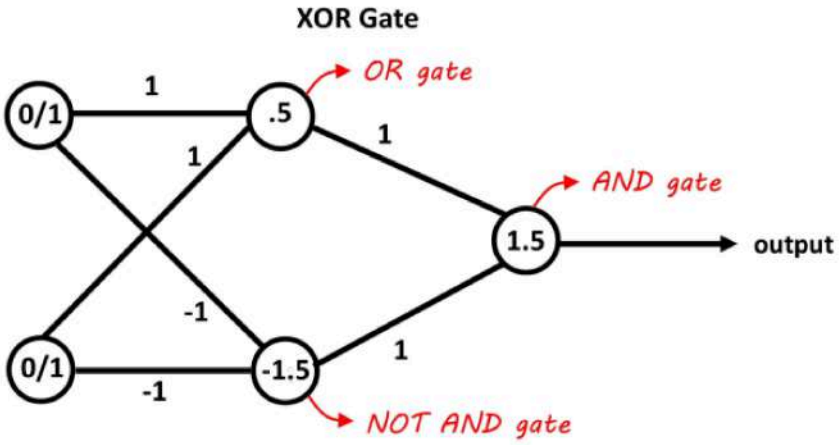
Figura 3 – Perceptron de multi camadas.



Fonte: Elipe (2020)

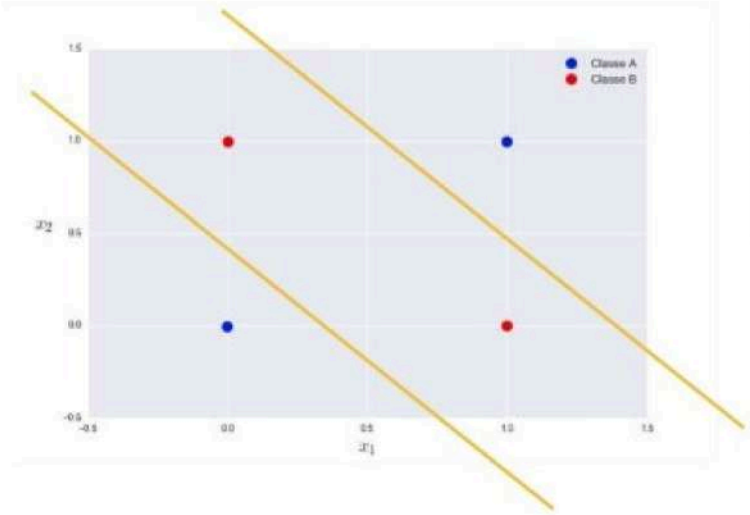
O caso do XOR pode ser solucionado com uma arquitetura semelhante (Figura 4), introduzindo dois neurônios em uma camada, onde cada neurônio é responsável pela separação linear das classes A e B (Figura 5), e uma camada extra para combinar os resultados individuais.

Figura 4 – Arquitetura do XOR.



Fonte: Neel (2015)

Figura 5 – XOR.



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Fonte: Neel (2015)

2.3.1 VETOR GRADIENTE.

Um conceito importante para o entendimento dos algoritmos de RNA é o vetor gradiente (∇), que indica em qual direção a função deve caminhar para atingir o seu valor máximo. Nas RNA busca-se minimizar a função de custo (erro) com o **gradiente descendente.**

Considere como exemplo a função $y = x^4 - 3x^3 + 2$ e o seu gráfico na Figura 6. Definimos o ponto aleatório [1] $x = -1{,}5$ e a sua derivada no ponto: (SOARES, 2018)

$$y'(-1{,}5) = 4 \times (-1{,}5)^3 - 9 \times (-1{,}5)^2 = -33{,}75 \tag{1}$$

Este valor representa o coeficiente angular da reta tangente naquele ponto, que nos dirá para aonde a função cresce se aumentar ou diminuir o valor de x

O novo valor de x, considerando uma taxa de aprendizado [2] de 0,0001:

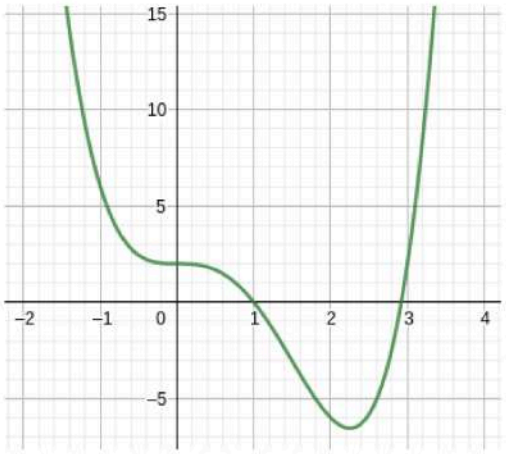
$$x_{novo} = (-1{,}5) - (0{,}0001) \times (33{,}75) \times (-1{,}5) = -1{,}4949 \tag{2}$$

A iniciação aleatória $x = 1{,}5$ será ajustada para $x = -1{,}4949$, após a primeira interação de ajuste dos pesos

[1] A definição do ponto é feita de forma aleatória pois em funções complexas não tem-se a ideia de onde estão os mínimos da função.

[2] A taxa de aprendizado é um hiper parâmetro importante pois definirá a velocidade e precisão de convergência do algoritmo.

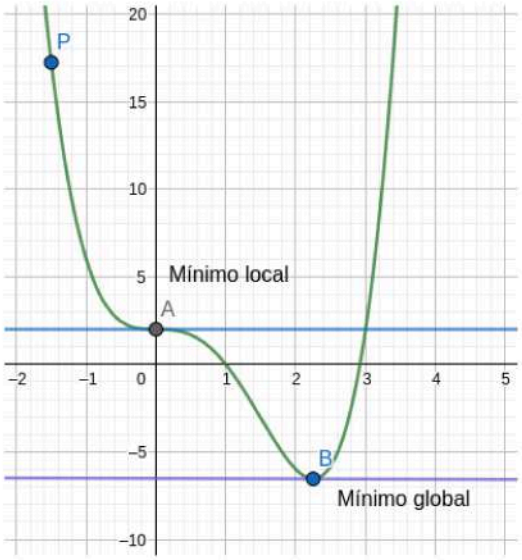
Figura 6 – Função: $y = x^4 - 3x^3 + 2$.



Fonte: Guimarães (2021)

O objetivo da descida do gradiente é alcançar o mínimo global da função, onde a função de erro será minimizada. A Figura 7 mostra que após o processo interativo de descidas de gradientes, o ponto P encontrará uma reta tangente igual a zero diferente do objetivo proposto, e o algoritmo ficará preso **preso no mínimo local**, mesmo em uma função simples como esta. A característica do problema irá informar se esta solução é adequada ou não. Este é um problema frequentemente encontrado em RNA, pois é um processo estocástico que busca soluções otimizadas subótimas. Essa característica reforça a inicialização de pesos de forma aleatória e o processo de treinamento e ajustes dos parâmetros de forma continua até se atingir os melhores resultados. Neste exemplo, uma pequena melhoria seria iniciar o $x \geq 2{,}5$ para encontrar o mínimo global com a descida do gradiente.

Figura 7 – Mínimo local.



Fonte: Guimarães (2021)

2.3.2 PROPAGAÇÃO (*FEEDFORWARD*).

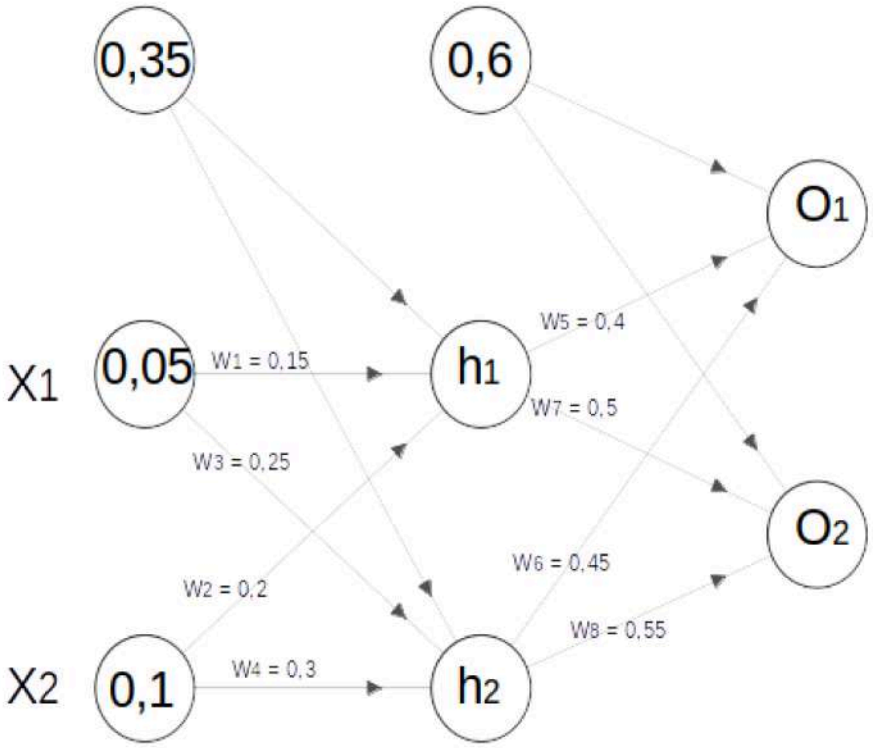
O MLP utiliza o algoritmo da propagação e retro propagação para atualização dos pesos. O *feedforward* são composições de funções simples que se propagam do começo da rede até o final, o exemplo[3] abaixo será utilizado para ilustrar o algoritmo. As entradas e saídas desejadas estão descritas na Tabela 1 e a arquitetura da rede do exemplo na Figura 8.

Tabela 1 – Entradas e saídas desejadas.

Entradas		Saídas desejadas	
x_1	x_2	y_1	y_2
0,05	0,1	0,01	0,99

Fonte: Soares (2018)

Figura 8 – Exemplo do algoritmo de propagação e retropropagação.



Fonte: Guimarães (2021)

- Os pesos W_1,W_2,..,W_7 foram iniciados de formas aleatórias.
- A função de custo foi definida como a função de erro quadrático (MSE)
- A função de ativação definida de cada LTU será a sigmoide. O principal conceito da função de ativação é introduzir não-linearidade ao sistema. A Tabela 2 demonstra esse conceito de não-linearidade

[3]Este exemplo foi retirado do curso de pós graduação da UFG (SOARES, 2018)

Propagando as entradas X_1 e X_2 pela camada 1

$$h_1 = (X_1 \times W_1) + (X_2 \times W_2) + (b1 \times 1) \tag{3}$$

$$h_1 = (0{,}05 \times 0{,}15) + (0{,}1 \times 0{,}02) + (0{,}35 \times 1) = 0{,}3775$$

$$h_2 = (X_1 \times W_3) + (X_2 \times W_4) + (b2 \times 1) \tag{4}$$

$$h_2 = (0{,}05 \times 0{,}25) + (0{,}1 \times 0{,}3) + (0{,}35 \times 1) = 0{,}39250$$

$$g(h_1) = \frac{1}{1 + \mathrm{e}^{-\mathrm{h_1}}} = \frac{1}{1 + \mathrm{e}^{-0{,}3775}} = 0{,}593269992 \tag{5}$$

$$g(h_2) = \frac{1}{1 + \mathrm{e}^{-\mathrm{h_2}}} = \frac{1}{1 + \mathrm{e}^{-0{,}39250}} = 0{,}596884378 \tag{6}$$

A camada 2 receberá os valores de Equação (5) e Equação (6) como entradas e propagará até a saída

$$o_1 = (g(h_1) \times W_5) + (g(h_2) \times W_6) + (b2 \times 1) = 1{,}105905967 \tag{7}$$

$$o_2 = (g(h_1) \times W_7) + (g(h_2) \times W_8) + (b2 \times 1) = 1{,}224921404 \tag{8}$$

$$g(o_1) = \frac{1}{1 + \mathrm{e}^{-\mathrm{o_1}}} = 0{,}751365070 \tag{9}$$

$$g(o_2) = \frac{1}{1 + \mathrm{e}^{-\mathrm{o_2}}} = 0{,}772928465 \tag{10}$$

2.3.3 RETRO PROPAGAÇÃO *(BACKPROPAGATION).*

O *backpropagation* é o algoritmo mais importante das redes neurais, foi introduzido em 1970, mas só ganhou destaque após um famoso *paper* de Rumelhart, Hinton e Williams (1986), que descreveu um novo procedimento de aprendizagem de retro propagação.

Observa-se que no exemplo de Soares (2018) os resultados da saída Equação (9) e Equação (10) são bastante diferentes do desejado, a aplicação do *backpropagation* é necessária para ajustar os pesos de forma que o gradiente estime a direção para cada peso de forma individual. A ideia do algoritmo é seguir o fluxo na direção contrária (do final para o começo da rede) aplicando o conceito da regra da cadeia Equação (11) e frações parciais, para decompor as funções compostas em funções mais simples e ter a relação de erro e peso.

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} \tag{11}$$

A função de erro quadrático é definida pela Equação (12), neste exemplo são dois erros:

$$E_{total} = \frac{1}{2}\sum_{k=1}^{N}(y_k - g(o_k))^2 = E_{o_1} + E_{o_2} \tag{12}$$

Os erros do neurônio 1 e 2 da camada de saída e o erro total são dados pelas equações 13, 14 e 15 respectivamente.

$$E_{o_1} = \frac{1}{2}(y_1 - g(o_1))^2 = \frac{1}{2}(0{,}01 - 0{,}751365070)^2 = 0{,}274811083 \tag{13}$$

$$E_{o_2} = \frac{1}{2}(y_2 - g(o_2))^2 = \frac{1}{2}(0{,}99 - 0{,}772928465)^2 = 0{,}023560026 \tag{14}$$

$$E_{total} = 0{,}023560026 + 0{,}023560026 = 0{,}298371109 \tag{15}$$

Aplicando o *backpropagation* apenas no W_5, é estimado o erro total em relação a W_5, segundo a regra da cadeia, aplica-se a Equação (16) ou a forma simplificada para algoritmo Equação (17).

$$\frac{\partial E_{total}}{\partial W_5} = \frac{\partial E_{total}}{\partial g(o_1)} \times \frac{\partial g(o_1)}{\partial o_1} \times \frac{\partial o_1}{\partial W_5} \tag{16}$$

$$\begin{aligned}\frac{\partial E_{total}}{\partial W_5} &= -(y1 - g(o_1)) \times g(o_1) \times (1 - g(o_1)) \times g(h_1) \\ &= 0{,}74136507 \times 0{,}186815602 \times 0{,}593269992 = 0{,}0822167041\end{aligned} \tag{17}$$

Por último, considerando a taxa de aprendizado = 0,5, o novo valor de W_5 é dado por:

$$W_5(t+1) = W_5(t) - (Learning\_rate) \times \frac{\partial E_{total}}{\partial W_5} = 0{,}4 - 0{,}5 * 0{,}0822167041 = 0{,}35891648 \tag{18}$$

O algoritmo segue o mesmo processo para atualização dos demais pesos, decompondo as funções complexas em parcelas simples.

Tabela 2 – Exemplo de introdução de não-linearidade com a função de ativação sigmoide.

$x \times \omega$	$\sigma(x \times \omega)$	$var(x \times \omega)$	$var(\sigma(x \times \omega))$
-2	0,12	-	-
-1,5	0,18	25%	50%
-1	0,27	33%	50%
-0,5	0,38	50%	41%
0	0,50	100%	32%
0,5	0,62	-	-
1	0,73	100%	18%
1,5	0,82	50%	12%
2	0,88	33%	7%

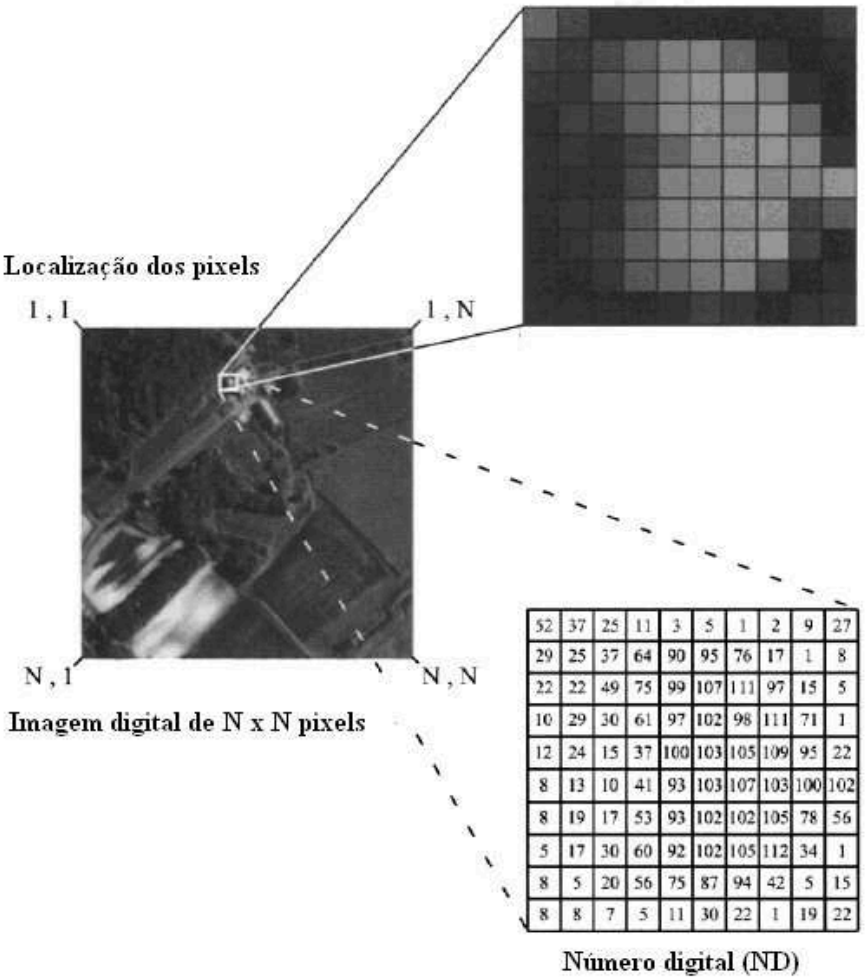
Fonte: Soares (2018)

2.4 REDES NEURAIS CONVOLUCIONAIS (CONVNET / CNN)

Lecun e Bengio (1995) apresentaram um algoritmo de aprendizagem profundo para trabalhar com visão computacional, onde os dados são de alta dimensão (imagens, vídeos, entre outros). As redes convolucionais (CNN) e o avanço da quantidade e qualidade dos dados foram fundamentais para o avanço científico do aprendizado profundo. Esta técnica é atualmente a mais utilizada no campo de visão computacional.

Uma imagem ou quadro de vídeo (imagem estática) é uma matriz de pixels, onde cada pixel tem a informação de cor do seu ponto na matriz (Figura 9). Em um arquitetura tradicional MLP, cada pixel seria uma entrada na RNA, por exemplo, uma imagem em tons de cinza com o número de pixel igual a $N = 256 \times 256$, teríamos N^2 entradas, em uma rede densa (totalmente conectada), adicionando uma camada oculta com o mesmo número de neurônios, teríamos N^4 sinapses (ligações). Neste simples exemplo de uma imagem em escala de cinza seriam 4 bilhões de pesos, o que torna a aplicação deste algoritmo inviável para dados densos.

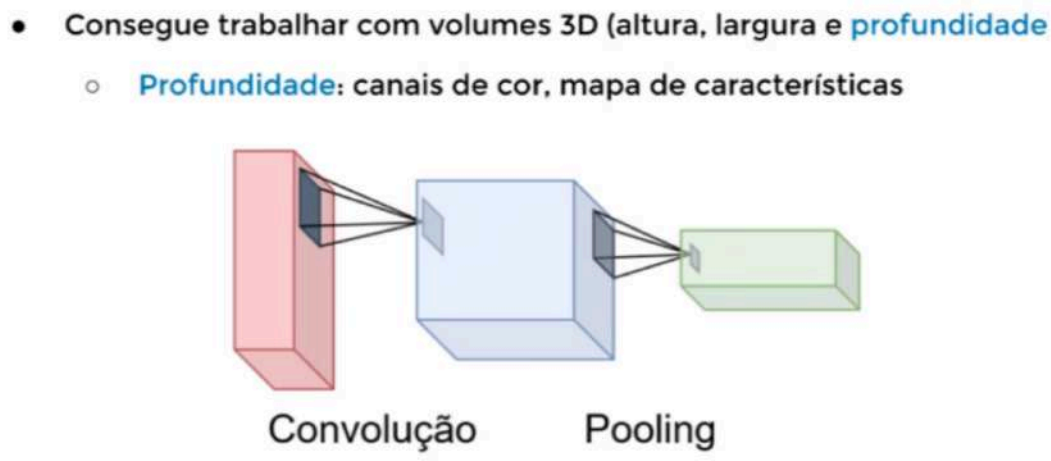
Figura 9 – Matriz de pixel da parte de uma imagem em tons de cinza.



Fonte: Elipe (2020)

O algoritmo de CNN é semelhante às MLP, a diferença fundamental é que cada unidade em uma camada CNN é um filtro 2D (Figura 10) que realiza um processo de convolução com a entrada e transforma a imagem em mapas de características. Isso é essencial para aprender padrões de dados de alta dimensão. Esta técnica funciona em problemas de aprendizagem supervisionada e não supervisionada: A aprendizagem supervisionada é aquele em que a entrada para o sistema e as saídas desejadas são conhecidas, e o modelo aprende um mapeamento com a resposta correta. No mecanismo de aprendizagem não supervisionado, os rótulos para um determinado conjunto de entradas não são conhecidos e o modelo estima a distribuição das amostras de dados de entrada (KHAN et al., 2018).

Figura 10 – Camadas básicas de uma CNN.



Fonte: Laranjeira (2018)

2.4.1 CAMADA CONVOLUCIONAL.

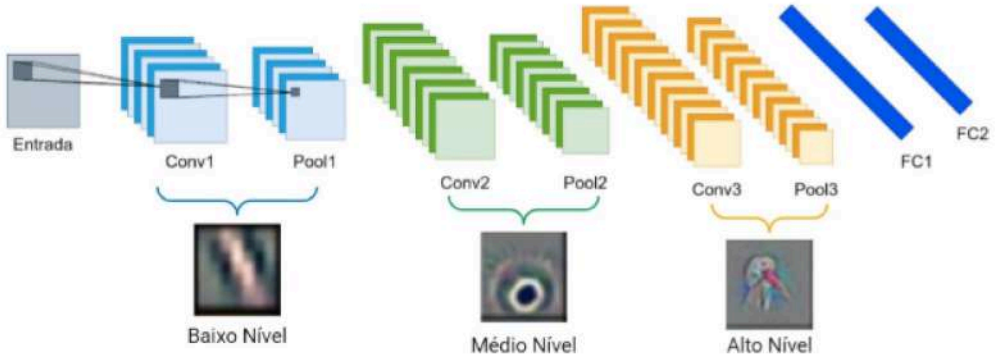
A camada convolucional é o componente mais importante da CNN, contém um conjunto de filtros (*kernels*) que realiza a operação de convolução com uma determinada entrada para gerar mapas de características. O filtro convolucional é uma matriz de números discretos e os seus pesos, de cada filtro, são aprendidos durante o treinamento da CNN. Uma das grande vantagens desse algoritmo é que a rede aprende automaticamente a extrair as melhores características da imagem, a partir da atualização dos pesos. (KHAN et al., 2018).

O aprendizado é de forma hierárquica (Figura 11) e as primeiras camadas de convolução são responsáveis por aprender características de baixo nível e as mais profundas de alto nível (LARANJEIRA, 2018).

2.4.1.1 PROCESSO DE CONVOLUÇÃO.

A Figura 12 exemplifica como é realizado o processo de convolução do *kernel* (verde) 2×2 varrendo (deslocamento) uma entrada de imagem 4×4 para produzir o mapa de

Figura 11 – Aprendizado hierárquico.



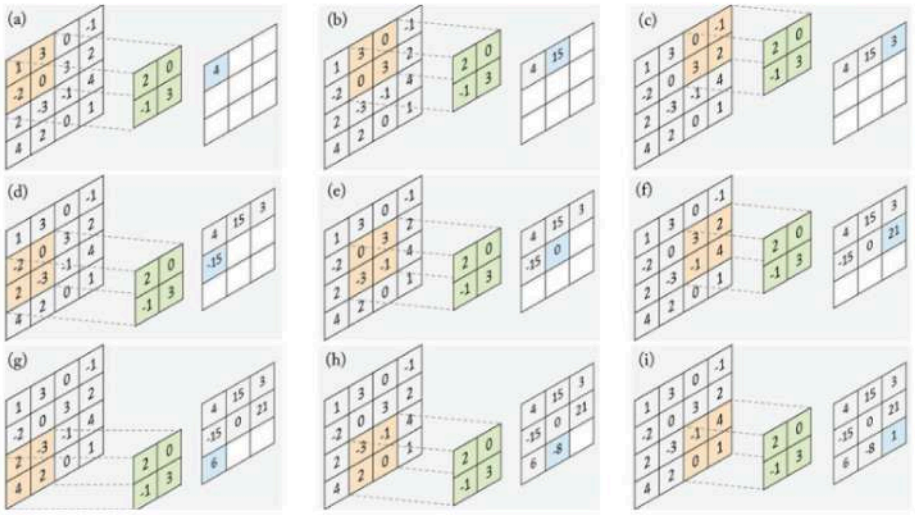
Fonte: Laranjeira (2018)

característica 3×3. O processo consiste nos seguintes passos:

$$img = \begin{bmatrix} 1 & 3 & 0 & -1 \\ -2 & 0 & 3 & 2 \\ 2 & -3 & -1 & 4 \\ 4 & 2 & 0 & 1 \end{bmatrix}, k = \begin{bmatrix} 2 & 0 \\ -1 & 3 \end{bmatrix}$$

1. Multiplicar $img(2 \times 2) \times k$ elementos da mesma posição.
2. Somar os elementos multiplicados e atribuir a posição n da matriz de saída.
3. Deslocar o filtro por toda a matriz com o valor do passo (*stride*)=1, neste caso, repetindo os passos 1 e 2.
4. O resultado final é um mapa de característica de menor dimensão 3×3 com as principais informações da imagem original. Essa redução nas dimensões fornece invariância moderada a escala e posição dos objetos.

Figura 12 – Exemplo de convolução.



Fonte: Khan et al. (2018)

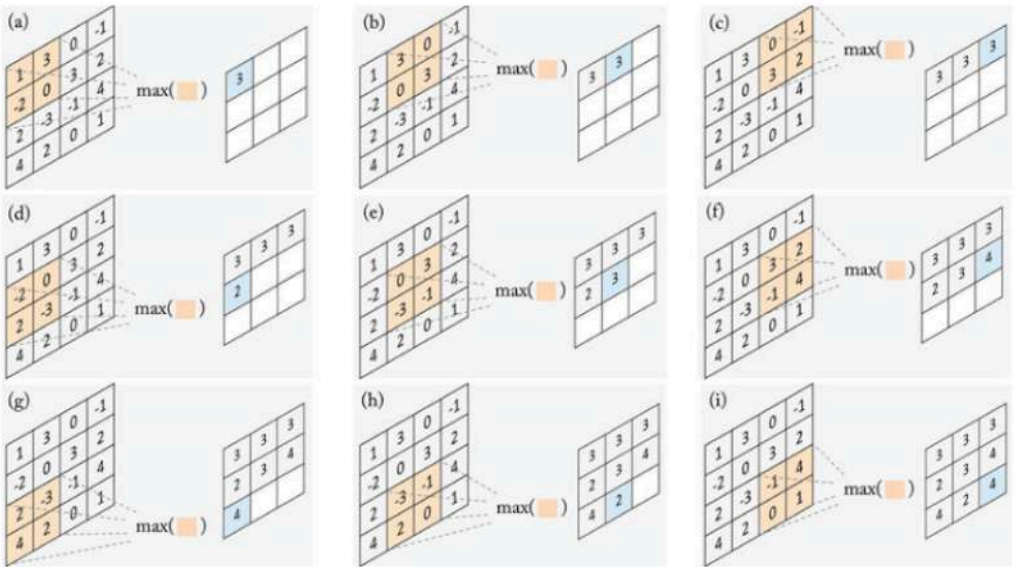
2.4.2 CAMADA DE AMOSTRAGEM (*POOLING*).

A camada de amostragem tem por objetivo realizar o *downsampling* e transformar a convolução invariante a translação, rotação e janelamento (*shifing*). Seu funcionamento consiste em receber o mapa de característica extraído pela camada de convolução de entrada e preparar um mapa de características condensado. Esta operação de combinação é definida por uma função de *pooling* (o mínimo, média ou o máximo). Para imagens é mais comumente utilizado o *max pooling*.

Semelhante à camada de convolução, é necessário especificar o tamanho da região agrupada e o passo. A Figura 13 mostra a operação de *max pooling*, onde é escolhido o valor máximo da matriz e o filtro *pooling* é deslizado sobre o mapa de característica.

A amostragem é útil para obter uma representação compacta de recursos que é invariante para mudanças moderadas na escala, pose e translação do objeto em uma imagem (GOODFELLOW; BENGIO; COURVILLE, 2016), além de reduzir o custo computacional, reduzindo o número de parâmetros a serem aprendidos.

Figura 13 – Exemplo de maxpooling



Fonte: Khan et al. (2018)

2.4.2.1 APRENDIZADO POR TRANSFERÊNCIA (*TRANSFER LEARNING*).

Uma das grandes vantagens do algoritmo de CNN é a possibilidade de reaproveitar a arquitetura e pesos (ou parte) de uma CNN pré-treinada e treinar a parte final do classificador para o ajustes dos pesos e modelo. Anualmente, novas arquiteturas são lançadas e avaliadas suas métricas em grandes *datasets*, como o ImageNet (DENG et al., 2009), que contem 1.000 classes de objetos e contém 1.281.167 imagens de treinamento, 50.000 imagens de validação e 100.000 imagens de teste.

3 ESTADO DA ARTE.

Baktir e Bolat (2020) propôs um sistema utilizando visão computacional com aprendizagem profunda avaliado no *dataset* PKLot. Foi utilizado uma rede base ResNet50 para aprendizagem por transferência e duas camadas ocultas com função de ativação softmax no classificador, obtendo 91,7% na base de validação e 99.82% na base de teste. Também foi realizado a validação em outras bases e o resultado foi apresentado no Quadro 1.

Quadro 1 – Resultados de Bakit e Bolat em duas bases de dados.

SET DE TREINO	SET DE TESTE	
	CNRPark	PKLot
PKLot	97,36%	99,82%
CNRPark	99,52%	99,28%

Fonte: Baktir e Bolat (2020)

Mettupally e Menon (2019) propuseram uma avaliação da precisão e tempo computacional de duas metodologias diferentes na base de dados do PKlot, uma rede CNN supervisionada, com apenas quatro camadas e uma biblioteca de detecção de objetos, Mask R-CNN, treinada no *dataset* COCO da Microsoft (HE et al., 2017). Os resultados foram detalhados no Quadro 2. Os autores avaliaram as métricas de tempo de detecção e optaram por uma arquitetura CNN simples e sem aprendizado por transferência

Quadro 2 – Comparação de CNN supervisionada e Mask R-CNN.

Modelo CNN	Precisão	Tempo computacional
4-Layer-CNN	87.69%	212.77s
Mask R-CNN	91,9%	57.03s

Fonte: Mettupally e Menon (2019)

Uma abordagem diferente de estacionamentos inteligentes foi trazida por Sieck, Calpin e Almalag (2020), sugerindo um método de estacionamento inteligente com visão computacional e um algoritmo de *Feature-based Cascade Classifier*, usado para detecção de objetos. As imagens são capturas e processadas em um NVIDIA Jetson Nano. Essas informações são alimentadas em um banco de dados MySQL e o aplicativo extrai os dados por meio de um API PHP. Seus resultados foram de 95% e 87% em dois bancos não especificados.

Dixit et al. (2020) forneceram um sistema utilizando tecnologias como android, internet das coisas (iot) e visão computacional. Em sua metodologia, um vídeo fornecido pelas câmeras CCTV é processado, atualizado em um banco de dados na nuvem e, simultaneamente, os dispositivos iot com sensores de proximidade validam se há carros no local. Este último trabalho

aborda técnicas diferentes, mais complexas, porém com maior confiabilidade, visto que utilizam *hardware* no lugar de um modelo preditivo de sistema de visão.

Os autores do *paper* (GöREN et al., 2019) sugerem um sistema de *smart parking* para estradas. As imagens capturadas das câmeras são armazenadas para treinar uma rede neural convolucional com algoritmos de *deep learning*. Por meio de uma solicitação via aplicativo a imagem atual da rua é capturada e processada para fornecer as vagas de estacionamento gratuitas na rua. O proposito deste *paper* é viabilizar o treinamento de uma rede neural profunda com uma pequena base de dados.

No *paper* (MEDURI; ESTEBANEZ, 2018) propõe uma revisão de algumas redes neurais convolucionais utilizadas para sistemas de *smart parking*. O objetivo deste trabalho é fazer um comparativo e fornecer dados de desempenho como precisão e eficiência para cada modelo. O quadro comparativo referencia vários autores entre 2016 e 2018 com seus modelos e as respectivas precisões para ambientes fechados e abertos.

4 METODOLOGIA

Um projeto de aprendizagem de máquinas é um processo cíclico até a otimização e que atenda os requisitos planejados. Em sua fase de projeto segue as seguintes etapas:

1. Coleta de dados.
2. Pré-processamento das imagens.
3. Escolha do modelo e definição da arquitetura.
4. Treinar o modelo.
5. Avaliar os resultados em bases de validação e teste.
6. Ajustar os parâmetros.

4.1 DELINEAMENTO DA PESQUISA.

Este trabalho é uma pesquisa aplicada que desenvolveu um método, equiparado a literatura, de análise de dados e automatiza a construção de modelos analíticos. Foi apresentado um modelo classificador de aprendizagem profunda para indicar a probabilidade de uma determinada vaga de estacionamento estar disponível ou ocupada.

A pesquisa foi realizada no período 15 de julho a 15 de outubro de 2021, com uma base de dados fornecida pela Universidade Federal do Paraná com amostras de 12.417 imagens em três ambientes.

4.2 COLETA DO BANCO DE DADOS.

A coleta de dados é uma das fases mais importantes em um projeto de aprendizagem profunda e um dos grandes responsáveis pelo crescimento exponencial deste algoritmo nas última década. Segundo Goodfellow, Bengio e Courville (2016), comparado com o aprendizado de máquina tradicional, o aprendizado profundo é mais adequado para o processamento de *big data*, pois o desempenho do algoritmo melhora com o aumento do volume de dados.

Este trabalho adotou um *dataset* brasileiro, criado por pesquisadores da Universidade Federal do Paraná e disponibilizado gratuitamente no kaggle[1]. Para o autor, "Um grande desafio para a realização de pesquisas envolvendo a classificação de vagas de estacionamento é a falta de conjunto de dados consistente e confiável. Para contornar esse problema, a principal contribuição deste *dataset* é a apresentar o PKLot, um robusto conjunto de dados de imagens de estacionamentos."(de Almeida et al., 2015)

[1] https://www.kaggle.com/blanderbuss/parking-lot-dataset

4.2.1 PKLOT - A ROBUST DATASET FOR PARKING LOT CLASSIFICATION.

- Contém 12.417 imagens dos estacionamentos e 695.899 ROI'S de vagas que foram manualmente checadas e rotuladas.
- Todas as imagens foram retiradas de estacionamentos da Universidade Federal do Parana (UFPR) e da Pontifícia Universidade Católica do Paraná (PUCPR), localizadas em Curitiba.
- O *dataset* é composto por três base dados diferentes: PUCPR, UFPR04 e UFPR05.

4.2.1.1 AQUISIÇÃO DAS IMAGENS.

As imagens foram coletadas em intervalos de 5 minutos, por mais de 30 dias, utilizando uma câmera de alta definição e baixo custo (Microsoft LifeCam HD-5000), posicionada no topo das construções.

O objetivo era obter capturas em três cenários distintos (nublado, ensolarado e chuvoso). As imagens da PUCPR foram capturadas do décimo andar, enquanto as bases de dados UFPR04 e UFPR05 foram coletadas na UFPR do quarto e do quinto andar, respectivamente (Figura 14).

Todas as imagens foram armazenadas em formato jpeg sem perda por compressão, em uma resolução de 1280×720.

Figura 14 – Exemplo de imagens do PKlot.

Fonte: de Almeida et al. (2015)

Tabela 3 – Resumo do *dataset* PKLOT

LOCAL	Condições	Dias	Img.	Ocupada	Vazia	Total
UFPR04	ENSOLARADO	20	2098	32166 (54.98%)	26334 (45.02%)	58400
	NUBLADO	15	1408	11608 (29.47%)	27779 (70.53%)	39387
	CHUVOSO	14	285	2351 (29.54%)	5607 (70.46%)	7958
	TOTAL	49	3791	46125 (43.58%)	59720 (56.42%)	105845
UFPR05	ENSOLARADO	25	2500	57584 (57.65%)	42306 (42.35%)	99890
	NUBLADO	19	1426	33764 (59.27%)	23202 (40.73%)	56966
	CHUVOSO	8	226	6078 (68.07%)	2851 (31.93%)	8929
	TOTAL	52	4152	97426 (58.77%)	68359 (41.23%)	165785
PUCPR	ENSOLARADO	24	2315	96762 (46.42%)	101672 (53.58%)	208433
	NUBLADO	11	1328	42363 (31.90%)	90417 (68.10%)	132780
	CHUVOSO	8	831	55104 (66.35%)	27951 (33.65%)	83056
	TOTAL	43	4474	194229 (45.78%)	230040 (51.46%)	424269

Fonte: de Almeida et al. (2015)

4.3 PRÉ-PROCESSAMENTO DOS DADOS.

As imagens deste conjunto de dados foram capturadas utilizando o mesmo dispositivo, com posição fixa e resolução de alta definição, desprezando ou minimizando a necessidade de alguma correção externa. As pastas são divididas de acordo com as condições climáticas e as sub pastas com a data.

4.3.0.1 ROTULAÇÃO.

O PKLOT fornece um arquivo (XML) de anotação (Figura 15) com os dados sobre cada imagem. A posição (X,Y) de cada vaga, o ângulo de rotação e se as vagas estão disponíveis ou ocupada.[2] Esta etapa de rotulação dos dados é essencial para o treinamento de um modelo com aprendizado supervisionado, onde o supervisor precisa informar a resposta correta para que a rede aprenda a mapear as características mais importantes e depois generalizar para outros dados nunca vistos. Este trabalho agrupou todas as fotos em uma única pasta para generalização dos dados.

4.3.0.2 SEGMENTAÇÃO DA ROI.

As ROI da imagem são demarcadas, conforme a Figura 16 e segmentadas em duas etapas distintas: para o treinamento do modelo e predição do classificador. O PKLOT disponibiliza as imagens segmentadas, mas a fim melhorar a segmentação e generalizar o projeto para outros cenários, foi desenvolvido um script para segmentação das ROI (Apêndice A) que segue a ideia do Algoritmo 2.

[2] O tensorflow, *framkework* de ML adotado neste projeto, possui recursos de *label* que identifica as classes vagas 'livres' e 'ocupadas' de acordo com divisão das subpastas.

Figura 15 – Exemplo de rotulação.

```
▼<parking id="ufpr04">
  ▼<space id="1" occupied="1">
    ▼<rotatedRect>
       <center x="724" y="623"/>
       <size w="93" h="154"/>
       <angle d="-31"/>
     </rotatedRect>
    ▼<contour>
       <point x="720" y="549"/>
       <point x="805" y="665"/>
       <point x="715" y="698"/>
       <point x="644" y="582"/>
     </contour>
   </space>
  ▼<space id="2" occupied="1">
    ▼<rotatedRect>
       <center x="774" y="511"/>
       <size w="79" h="132"/>
       <angle d="-42"/>
     </rotatedRect>
    ▼<contour>
       <point x="769" y="451"/>
       <point x="849" y="534"/>
       <point x="789" y="586"/>
       <point x="701" y="489"/>
     </contour>
```

Fonte: Guimarães (2021)

Algoritmo 2: Geração de ROI

Input: IMG,XML
Output: ROI

1. Carregar IMG e XML correspondente.
2. Filtrar os campos de interesses (vagas e *label*) no XML.
3. Demarcar e segmentar cada ROI.
4. Renomear ROI - vagaX_ocupadoX_img_X
5. Salvar as ROI em pasta de acordo com o label (livre,ocupada)

O intervalo de 5 minutos nas capturas incrementou muitas imagens semelhantes ao modelo (Figura 17), decidiu-se realizar uma nova segmentação com um passo de meia hora para cada foto, resultando na tabela Tabela 4.

Tabela 4 – Tabela das ROI'S

LOCAL	Qtd. imagens	Vagas Ocupada	Vagas Vazia	Total
UFPR04	3791	5336	5248	58400
UFPR05	4152	16292	11388	27680
PUCPR	4474	21598	33302	54900

Fonte: Guimarães (2021)

Figura 16 – ROI demarcadas.

Fonte: Guimarães (2021)

Figura 17 – ROI segmentadas.

Fonte: Guimarães (2021)

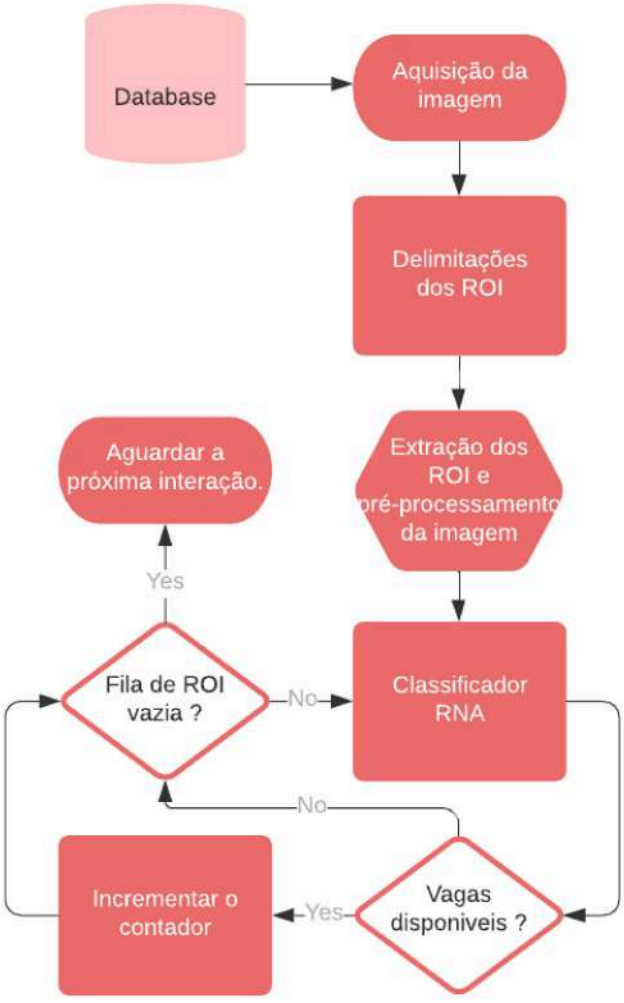
4.4 MODELO.

4.4.1 OBJETIVO DO SISTEMA.

O fluxograma da Figura 18 descreve o processo do sistema:

1. Carregar uma imagem do banco de dados em um tempo pré-determinado.
2. Delimitar as ROI.
3. Pré-processar a imagem e segmentar as ROI.
4. Converter em tensores e alimentar o classificador.
5. Fazer a predição no classificador.
6. Resposta maior que limiar definido, incrementar o contador de vagas livres.
7. Puxar uma nova ROI da pilha, se disponível, e repetir os passos 5, 6 e 7.
8. Caso a pilha esteja vazia, zerar o contador e aguardar a próxima interação.

Figura 18 – Fluxograma do sistema.



Fonte: Guimarães (2021)

4.4.2 FERRAMENTAS.

O Python é a linguagem de programação mais adotada pelos cientistas de dados para modelos de ML, os principais motivos são a facilidade de codificação e a grande quantidade de *frameworks* e bibliotecas disponíveis. Apesar de ser uma linguagem interpretada, é utilizada em sua maioria das vezes como interface, no treinamento e predição do modelo os *frameworks* utilizam a API do CUDA em baixo nível para gerenciar os dados na memoria da GPU (Figura 19).

Figura 19 – Ferramentas Python.

Fonte: Kalbande (2020)

Para o desenvolvimento do projeto foram utilizadas as seguintes *API,frameworks e libs*, e suas respectivas referências foram recorrentes buscas de estudos.

- **Tensorflow** - "O TensorFlow é uma plataforma completa de código aberto para machine learning. Ele tem um ecossistema abrangente e flexível de ferramentas, bibliotecas e recursos da comunidade que permite aos pesquisadores levar adiante ML de última geração e aos desenvolvedores criar e implantar aplicativos com tecnologia de ML"(ABADI et al., 2015).
- **Keras** - "Keras é uma API projetada para seres humanos, não máquinas. Keras segue as práticas recomendadas para reduzir a carga cognitiva: oferece APIs consistentes e simples, minimiza o número de ações do usuário necessárias para casos de uso comuns e fornece mensagens de erro claras e acionáveis. Ele também possui uma extensa documentação e guias do desenvolvedor"(CHOLLET et al., 2015).
- **Numpy** - "NumPy é o pacote fundamental para computação científica em Python. É uma biblioteca Python que fornece um objeto de matriz multidimensional, vários objetos derivados (como matrizes e matrizes mascaradas) e uma variedade de rotinas para operações rápidas em matrizes, incluindo matemática, lógica, manipulação de forma,

classificação, seleção, E / S , transformadas discretas de Fourier, álgebra linear básica, operações estatísticas básicas, simulação aleatória e muito mais"(HARRIS et al., 2020).

- **Pandas** - "O pandas é uma ferramenta de análise e manipulação de dados de código aberto rápida, poderosa, flexível e fácil de usar, construído com base na linguagem de programação Python"(TEAM, 2020).
- **Opencv** - "OpenCV (Open Source Computer Vision Library) é uma biblioteca de software de visão computacional e aprendizado de máquina de código aberto. O OpenCV foi construído para fornecer uma infraestrutura comum para aplicativos de visão computacional e para acelerar o uso da percepção da máquina em produtos comerciais"(BRADSKI, 2000).
- **Matplotlib** - "Matplotlib é uma biblioteca abrangente para a criação de visualizações estáticas, animadas e interativas em Python. capturas de tela. Matplotlib torna as coisas fáceis mais fáceis"(HUNTER, 2007).
- **Jupyter Notebook** - "JupyterLab é um ambiente de desenvolvimento interativo baseado na web para notebooks, código e dados Jupyter. O JupyterLab é flexível: configure e organize a interface do usuário para oferecer suporte a uma ampla variedade de fluxos de trabalho em ciência de dados, computação científica e aprendizado de máquina"(KLUYVER et al., 2016).

4.4.3 TREINAMENTO.

4.4.3.1 CAMADA DE ENTRADA.

O banco de dados escolhidos para o treinamento foi o UFPR05, com ROI's de tamanho 160x160 RGB e a divisão do banco foi de 70% para treinamento e 30% para validação e teste[3]

4.4.3.2 APRENDIZADO POR TRANSFERÊNCIA.

A arquitetura deste trabalho (Figura 20) teve o aprendizado por transferência da MobileNetV2, que tem uma arquitetura leve ideal para projetos *mobile* e é descrita detalhadamente no *paper* de Sandler et al. (2018). É utilizado um extrator de característica 5x5 fornecido MobileNetV2, com pesos pré-treinado no banco da ImageNet e adicionado ao sistema (Figura 21).

[3]Entre os 30%, 80% para validação e 20% para teste.

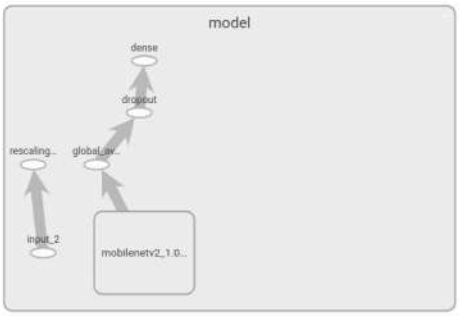
Figura 20 – Camadas.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 160, 160, 3)]	0
rescaling_1 (Rescaling)	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Functi	(None, 5, 5, 1280)	2257984
global_average_pooling2d (Gl	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 1)	1281

Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984

Fonte: Guimarães (2021)

Figura 21 – Fluxo.



Fonte: Guimarães (2021)

4.4.3.3 HIPER PARÂMETROS (*HYPERPARAMETERS*).

- Número de camadas densas = 1.
- *Dropout* = 20%. [4]
- Função de ativação - sigmoide.
- Função de perda - *BinaryCrossentropy* (Utilizada para problemas de classificação).
- Taxa de aprendizado = 10^{-4}.
- Número de épocas = 10.
- Tamanho do lote (*batch size*) = 32. [5]

[4] *Dropout* é uma técnica de regularização onde aleatoriamente exclui um percentual de entradas e neurônios, esta técnica é usada para evitar que os dados se ajustem demasiadamente a base de treinamento (*overfitting*) e caracterize pequena capacidade de generalização dos dados.

[5] É um hiper parâmetro importante que define a quantidade de amostras que serão treinadas antes da atualização dos pesos. O tamanho do lote pode evitar mínimos locais e garantir a convergência.

5 CRONOGRAMA.

O cronograma do projeto foi realizado conforme o Quadro 3.

Quadro 3 – Cronograma do projeto.

Atividades	**Semana 1**	**Semana 2**	**Semana 3**	**Semana 4**
Estudar ML.	X	X	X	X
Estudar Python.	X	X	X	X
Idealizar projeto.	X	X		
Estudar artigos recentes.		X	X	
Aquisição de *datasets*				X
	Semana 5	**Semana 6**	**Semana 7**	**Semana 8**
Estudar RNA E CNN.	X	X	X	X
Estudar Python.	X	X	X	X
Estudar *frameworks para dp*.	X	X	X	X
Tratar imagens	X	X		X
Modelar rede.		X	X	
Avaliar modelo.				X
Retratar imagens.				X
	Semana 9	**Semana 10**	**Semana 11**	**Semana 12**
Estudar Python.	X	X	X	X
Adicionar *transfer learning*	X			
Avaliar modelo e resultados		X	X	X
Escrever TCC			X	X

Fonte: Guimarães (2021)

6 ANÁLISE E DISCUSSÃO DOS RESULTADOS

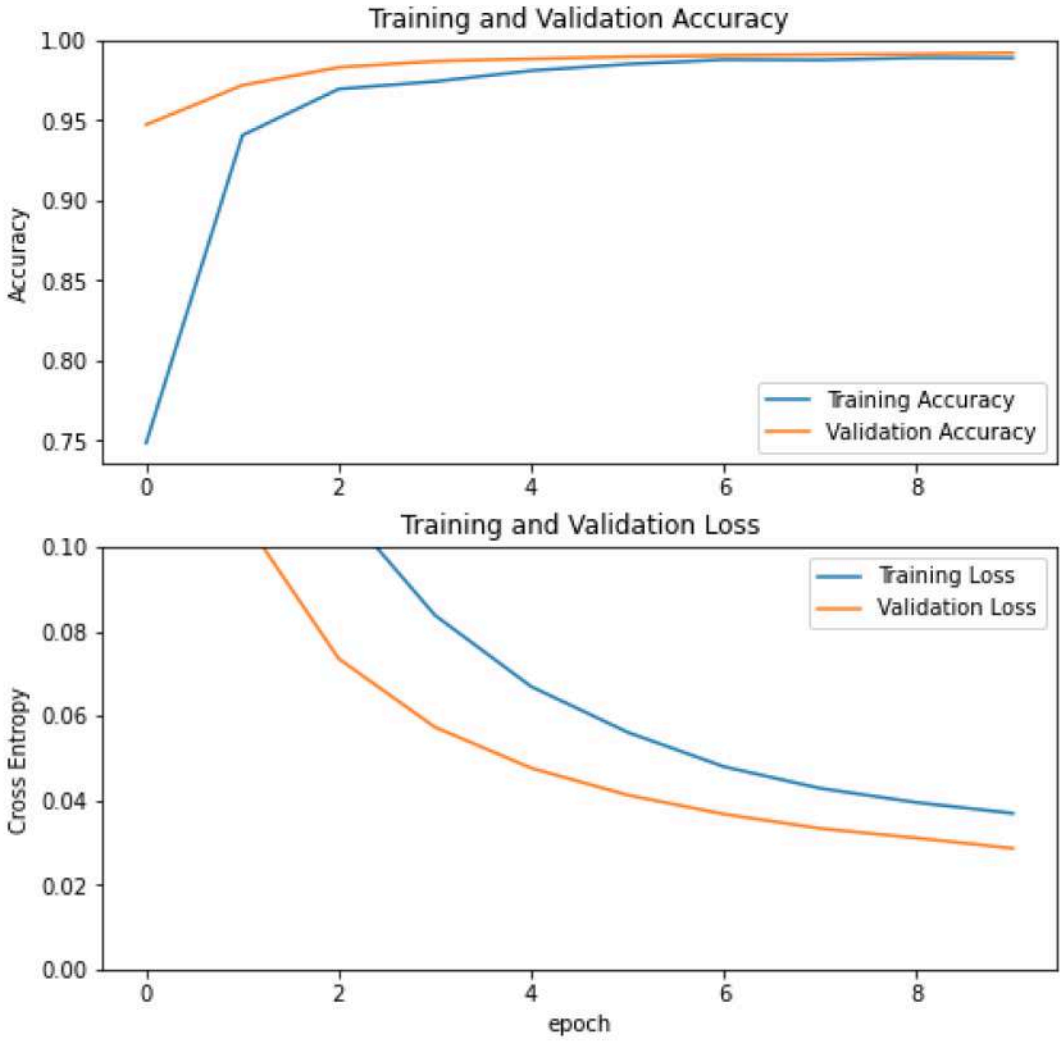
As métricas desses experimentos estão disponíveis na plataforma do *tensorboard* por meio dos links: UFPR04, UFPR05, PUCPR. Foram avaliadas as métricas de precisão e perdas durante o treinamento nas divisões de treino e validação e avaliado na divisão de teste. A principio não foi realizado o *cross* dos bancos de dados.

6.1 RESULTADOS.

6.1.0.1 UFPR04

O Quadro 4 e as Figura 22 e Figura 23 ilustram o desempenho do UFPR04.

Figura 22 – Precisão e perdas por época - UFPR04.



Fonte: Guimarães (2021)

Quadro 4 – Resumo do treinamento UFPR04.

UFPR04 - 28 vagas	Precisão	Perda
Avaliação na base de treino	0,9902	0,0388
Avaliação na base de validação	0,9907	0,0410
Avaliação na base de teste	0,9918	0,0441
Épocas	10	
Lote	32	
Taxa de aprendizado	0,0001	
Dropout	20%	
Otimizador	Adam	
Função de Perda	*BinaryCrossentropy*	
Métricas	Precisão	

Fonte: Guimarães (2021)

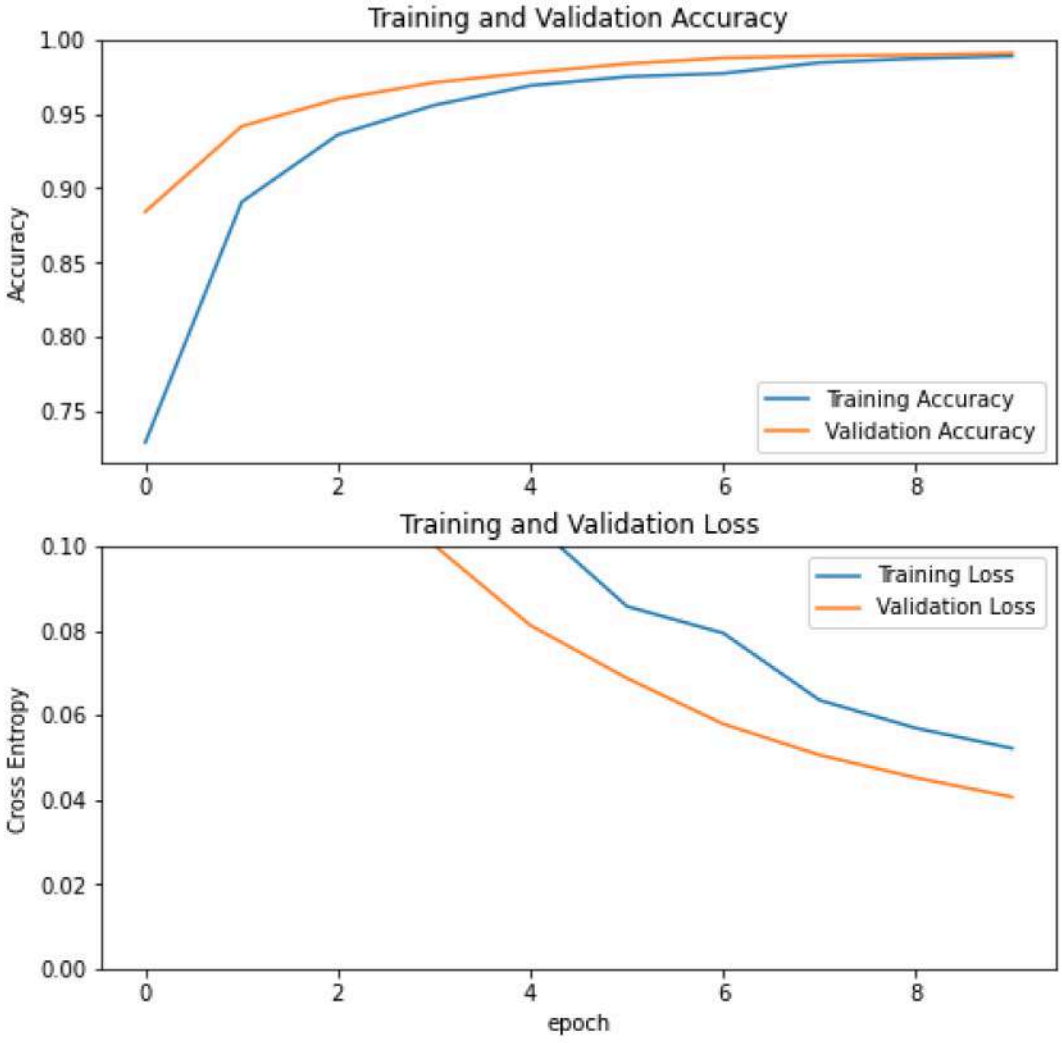
Figura 23 – Predição em lote - UFPR04.

Fonte: Guimarães (2021)

6.1.0.2 UFPR05

O Quadro 5 e as Figura 24 e Figura 25 ilustram o desempenho do UFPR05.

Figura 24 – Precisão e perdas por época - UFPR05.



Fonte: Guimarães (2021)

Quadro 5 – Resumo do treinamento UFPR05.

UFPR05 - 37 vagas	Precisão	Perda
Avaliação na base de treino	0,9922	0,0287
Avaliação na base de validação	0,9923	0,0286
Avaliação na base de teste	0,9920	0,0285
Épocas	10	
Lote	32	
Taxa de aprendizado	0,0001	
Dropout	20%	
Otimizador	Adam	
Função de Perda	*BinaryCrossentropy*	
Métricas	Precisão	

Fonte: Guimarães (2021)

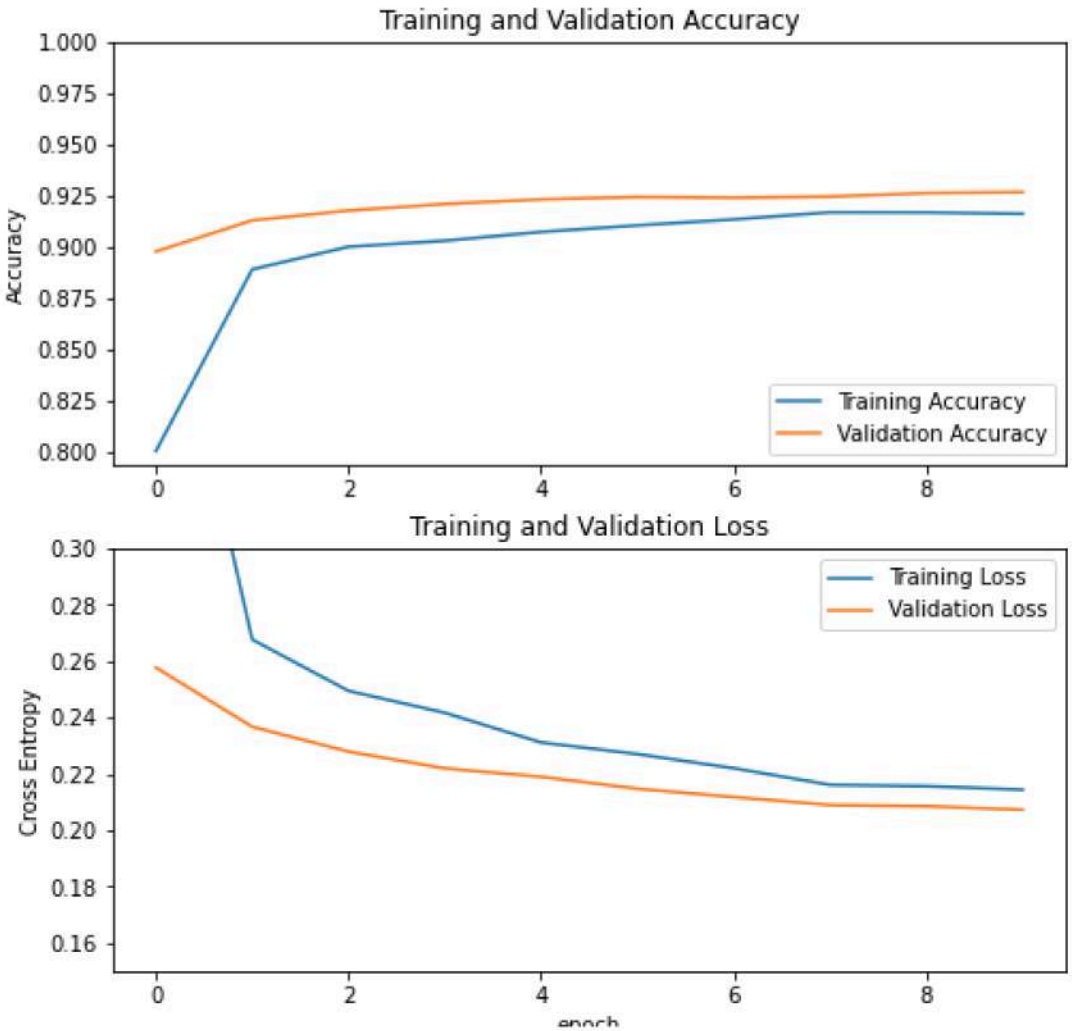
Figura 25 – Predição em lote - UFPR05.

Fonte: Guimarães (2021)

6.1.0.3 PUCPR

O Quadro 6 e as Figura 26 e Figura 27 mostram o desempenho do PUCPR.

Figura 26 – Precisão e perdas por época - PUCPR.



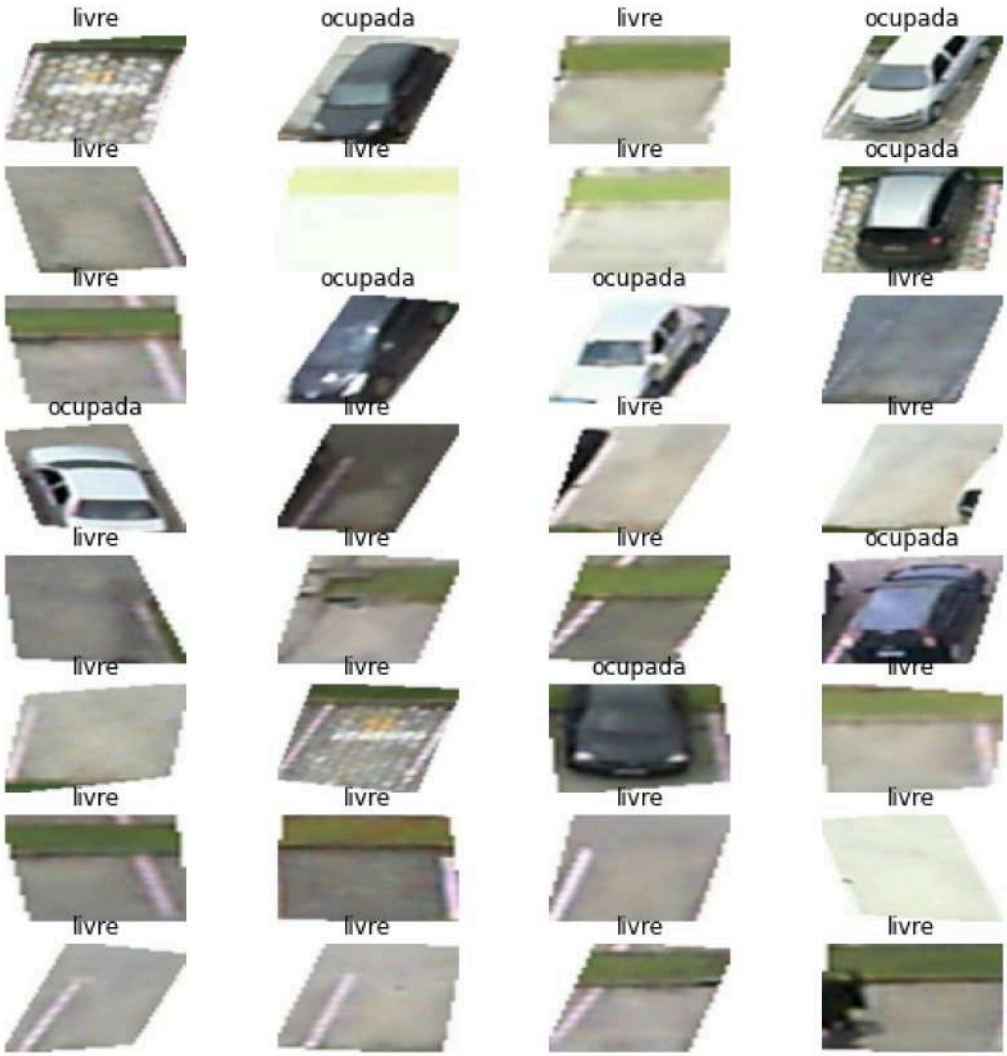
Fonte: Guimarães (2021)

Quadro 6 – Resumo do treinamento PUCPR.

PUCPR - 100 vagas	Precisão	Perda
Avaliação na base de treino	0,9313	0,1976
Avaliação na base de validação	0,9267	0,2069
Avaliação na base de teste	0,9267	0,2061
Épocas	10	
Lote	32	
Taxa de aprendizado	0,0001	
Dropout	20%	
Otimizador	Adam	
Função de Perda	*BinaryCrossentropy*	
Métricas	Precisão	

Fonte: Guimarães (2021)

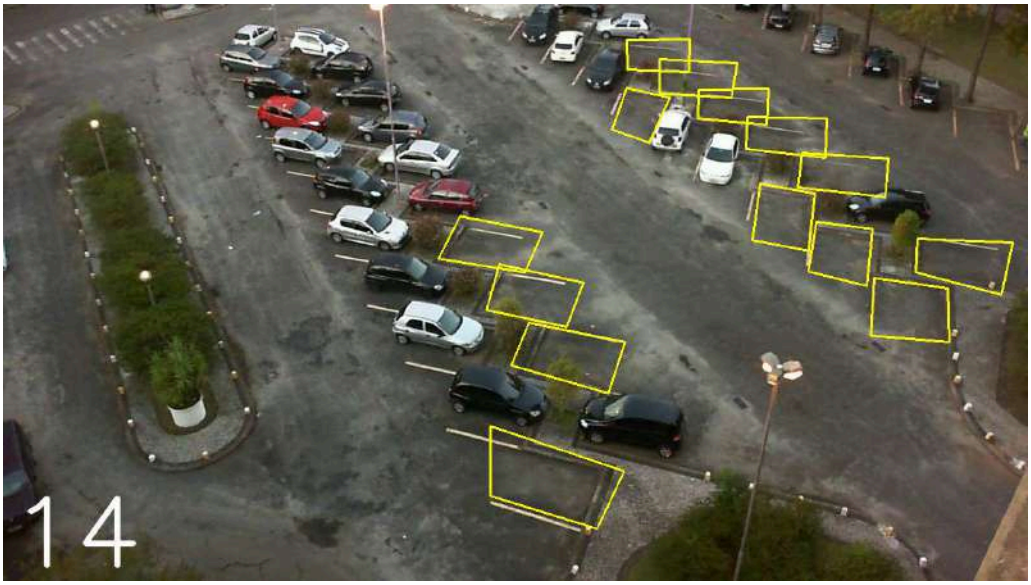
Figura 27 – Predição em lote - PUCPR.



Fonte: Guimarães (2021)

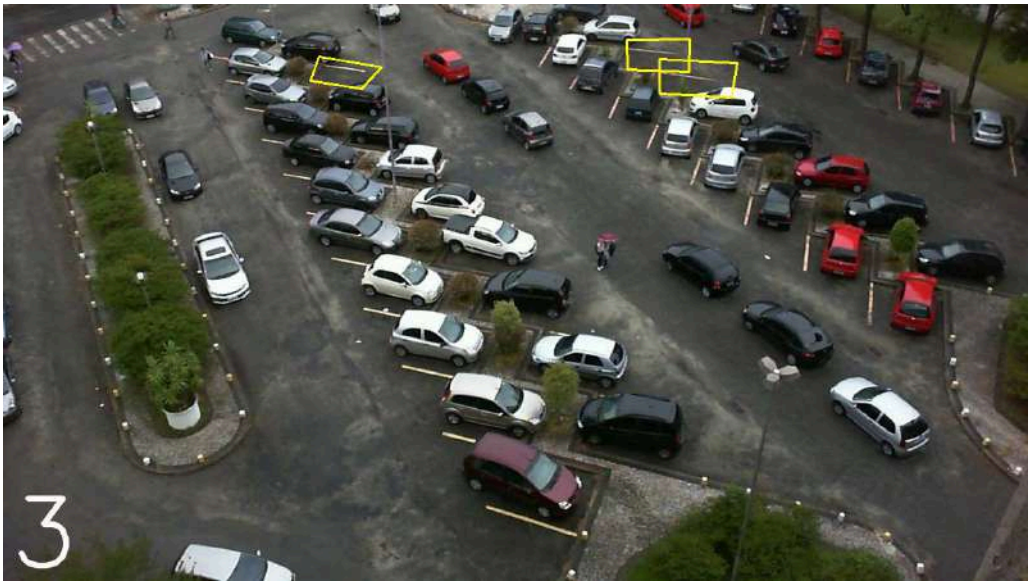
6.2 RESULTADOS PRÁTICOS.

Figura 28 – Predição do sistema em tempo real.



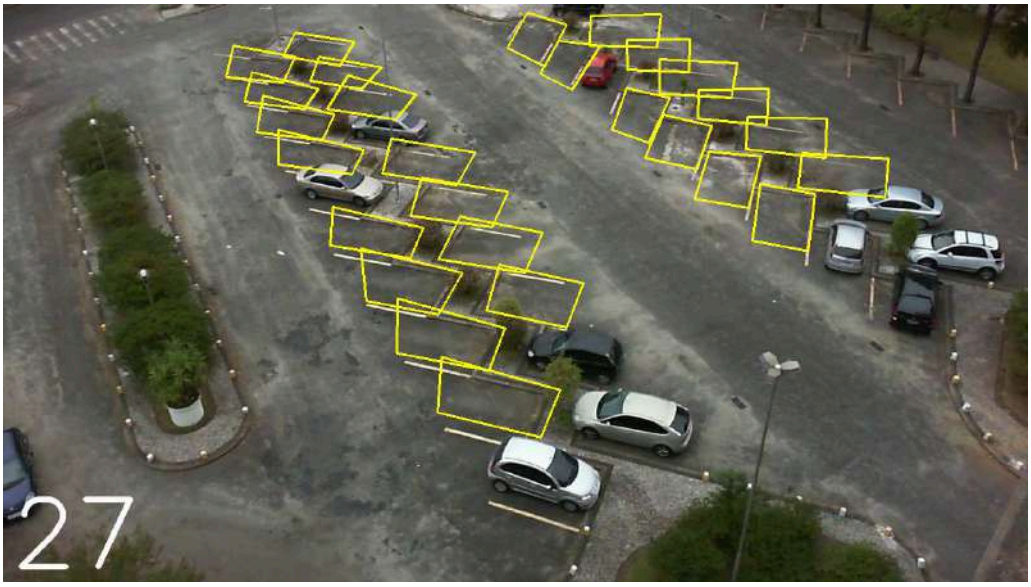
Fonte: Guimarães (2021)

Figura 29 – Predição do sistema em tempo real.



Fonte: Guimarães (2021)

Figura 30 – Predição do sistema em tempo real.

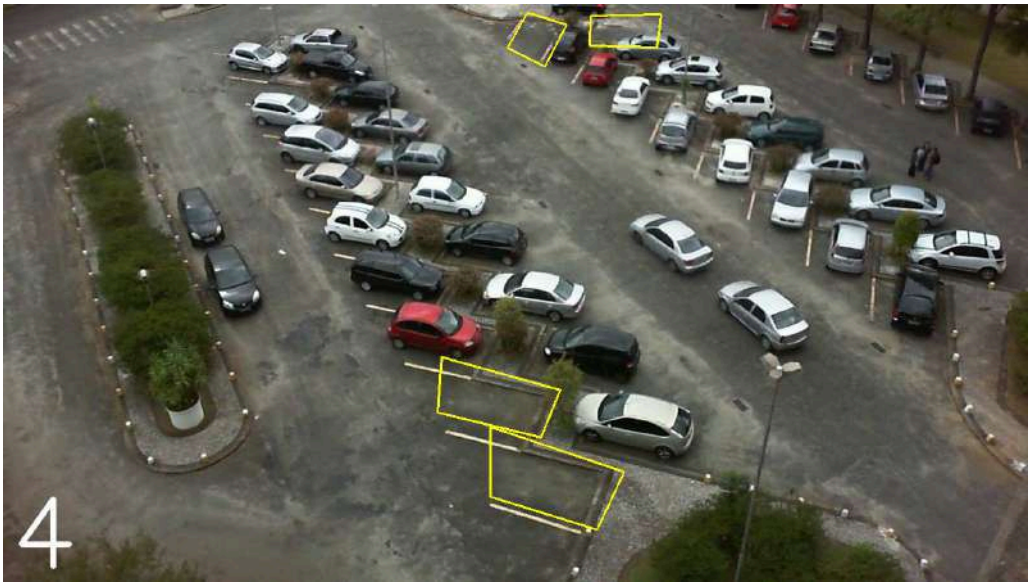


Fonte: Guimarães (2021)

Figura 31 – Predição do sistema em tempo real.

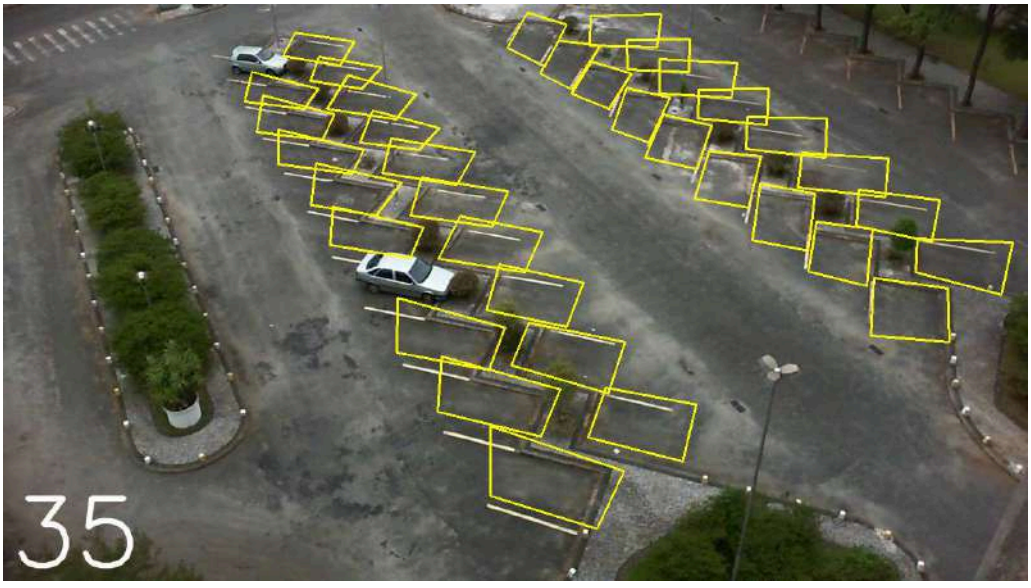
Fonte: Guimarães (2021)

Figura 32 – Predição do sistema em tempo real.



Fonte: Guimarães (2021)

Figura 33 – Predição do sistema em tempo real.

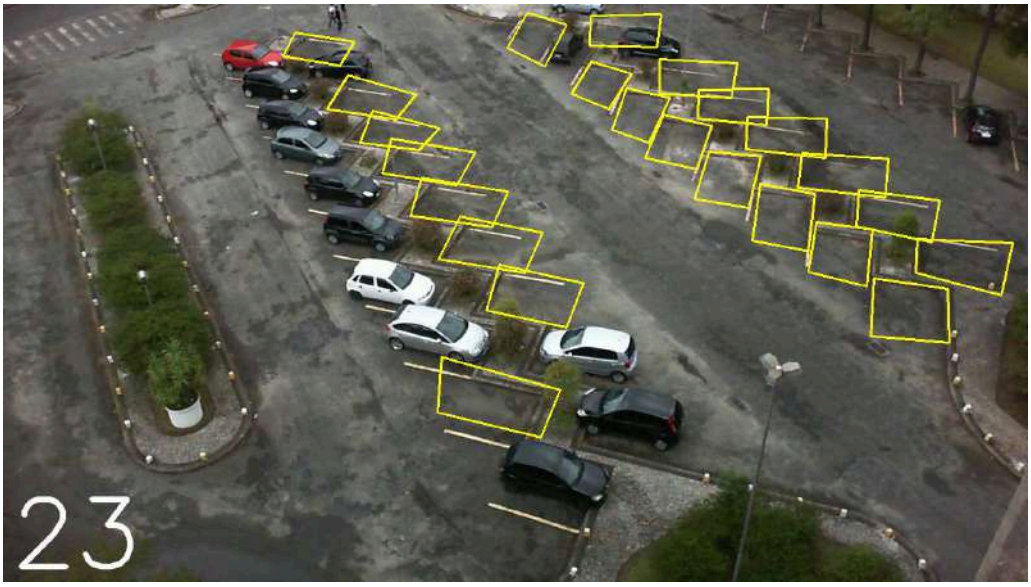


Fonte: Guimarães (2021)

Figura 34 – Predição do sistema em tempo real.

Fonte: Guimarães (2021)

Figura 35 – Predição do sistema em tempo real.



Fonte: Guimarães (2021)

6.3 DISCUSSÃO.

Os resultados apresentados neste trabalho foram satisfatórios e o projeto do classificador (Apêndice B) respondeu bem nos testes com as imagens aleatórias do estacionamento completo (todas as vagas), carregadas na base da dados do UFPR05. A predição das 37 vagas ocorre quase de imediato, com pouquíssimo erro, avaliando de forma individual, em tempo real e sem média. Todavia, no vídeo com 15 amostras por segundo, o sistema apresentou um pequeno

delay, necessitando aplicar técnicas de paralelismo para um melhor desempenho, se necessário, visto que a proposta de estacionamentos inteligentes em sua maior parte das aplicações será um sistema de dinâmica lenta.

Essa etapa inicial ainda precisa de continuação e ser avaliada em outros bancos para analisar questões importantes como *overfiting*, outras arquiteturas e um ajuste fino dos parâmetros, que serão detectados na fase de melhorias do modelo, implementação e monitoramento. O ciclo de um projeto de ML é bastante recursivo e mesmo após obter a melhor sintonia precisa ser monitorado ao longo da sua vida útil.

Os melhores resultados foram obtidos na base de dados das imagens capturas no UFPR05, devido a um melhor campo de visão da câmera e uma rotulagem mais detalhada das anotações de cada espaço de vaga, este também foi o banco escolhido para a construção do sistema e predição em tempo real (Figura 28). Na base de dados do UFPR04, notou-se problemas durante a captura das imagens, sendo necessário o reposicionamento da câmera, que resultou em pontos de segmentação das anotações diferentes e fora de alinhamento em diversas imagens. Enquanto, no PUCPR a distância da câmera afetou a qualidade das imagens e provavelmente é o motivo de ter poucas vagas segmentadas proporcional ao seu número. Para melhores resultados nesses dois bancos é preciso refazer os pontos de segmentações dos arquivos de anotações (XML).

Os autores Baktir e Bolat (2020) corroboram a utilização da metodologia adotada neste trabalho. Além disso, forneceram preciosas futuras melhorias para o projeto, onde buscou-se avaliar o desempenho em outras bases de dados, como o CNRPark. Contudo, os pesquisadores utilizaram uma arquitetura base mais pesada e antiga comparada à MobileNetV2, adotada neste trabalho, que é mais indicada para as pretensões em construção para dispositivos móveis.

Mettupally e Menon (2019) reforça a necessidade de uma CNN mais robusta para obter melhores acurácia e a métrica de tempo não é um fator crucial, visto que estacionamentos inteligentes são sistemas de inércia lenta. Não foi descrito com detalhes a forma de predição, mas os resultados de tempo computacional foram superiores ao deste trabalho, na mesma base PUCPR.

Sieck, Calpin e Almalag (2020) e Dixit et al. (2020) trouxe uma valiosa contribuições sobre a possibilidade de utilizar um microprocessador para fazer as predições dos modelos, dispensando o uso de um servidor, e uma interface de API PHP para os dispositivos móveis. O projeto de Dixit et al. (2020) também fornece a possibilidade de utilizar sensores IOT para ter predições mais robustas. Porém essa ideia não está nos planos atuais porquê aumentaria o orçamento e a complixidade do sistema, praticamente inviabilizando para projetos em áreas públicas.

7 CONCLUSÃO

Projetos na área de inteligência artificial, especialmente de aprendizagem profunda para sistemas de visão, tem recebido uma enorme visibilidade nos últimos anos, consequência do enorme volume de dados e poder computacional da atualidade. Os algoritmos de CNN são beneficiados com uma quantidade relevante de dados porque aprendem padrões e extraem mapas de características, reduzindo significamente a densidade e complexidade dos dados.

No setor de cidades e estacionamentos inteligentes não é diferente, muitas pesquisas e técnicas na área de IA estão sendo discutidas para propor uma solução eficiente para um problema inevitável de mobilidade urbana, financeiro e social, já que o trânsito é um dos grandes responsáveis pelo estresse pessoal.

Este trabalho apresentou um modelo computacional de predição de vagas segmentadas de estacionamentos e um sistema em tempo real para validação do modelo. Obteve-se resultados promissores para um projeto inicial e a metodologia é equiparada as pesquisas da comunidade científica relatado em artigos pelo mundo. Contudo, o modelo ainda precisa ser validado em diferentes bases de dados para avaliar a sua capacidade de generalização a outros dados e extrair outras métricas importantes, como *overfitting*.

7.1 TRABALHOS FUTUROS

- Validar os resultados em novas bases e extrair novas métricas.
- Avaliar a melhor arquitetura.
- Adicionar as etapas de implementações e monitoramento.
- Sintonizar os hiper parâmetros do modelo.
- Criar interface móvel para o público alvo.
- Publicar artigos para disseminar o conhecimento cientifico obtido.

7.2 CONSIDERAÇÕES FINAIS

Este trabalho apresentado possibilita reflexões, colaborações nos resultados e metodologia e está suscetível a novas revisões.

Referências

ABADI, M. et al. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>. Citado na página 23.

ARNOTT, R.; INCI, E. An integrated model of downtown parking and traffic congestion. **Urban Economics**, v. 60, n. 3, p. 418–442, 2006. Citado na página 1.

AZEVEDO, S.; RIBEIRO, L. **Mapa da motorização individual no Brasil**. 2019. Disponível em: <https://www.observatoriodasmetropoles.net.br/wp-content/uploads/2019/09/mapa_moto2019v2.pdf>. Acesso em: 28 de julho de 2021. Citado na página 1.

BAKTIR, A. B.; BOLAT, B. Determining the occupancy of vehicle parking areas by deep learning. In: **2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)**. [S.l.: s.n.], 2020. p. 1–4. Citado 2 vezes nas páginas 15 e 37.

BRADSKI, G. The OpenCV Library. **Dr. Dobb's Journal of Software Tools**, 2000. Citado na página 24.

CHOLLET, F. et al. **Keras**. GitHub, 2015. Disponível em: <https://github.com/fchollet/keras>. Citado na página 23.

de Almeida, P. R. et al. Pklot – a robust dataset for parking lot classification. **Expert Systems with Applications**, v. 42, n. 11, p. 4937–4949, 2015. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417415001086>. Citado 3 vezes nas páginas 17, 18 e 19.

DEMUSH, R. **https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3**. 2019. Disponível em: <https://www.observatoriodasmetropoles.net.br/wp-content/uploads/2019/09/mapa_moto2019v2.pdf>. Acesso em: 29 de setembro de 2021. Citado na página 2.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: IEEE. **2009 IEEE conference on computer vision and pattern recognition**. [S.l.], 2009. p. 248–255. Citado na página 14.

DIXIT, M. et al. Smart parking with computer vision and iot technology. p. 170–174, 2020. Citado 2 vezes nas páginas 15 e 37.

ELIPE, D. R. **Rede Perceptron de uma única camada**. 2020. Disponível em: <https://github.com/d-r-e/multilayer-perceptron>. Acesso em: 05 de outubro de 2021. Citado 2 vezes nas páginas 5 e 11.

FGV. **O que é uma cidade inteligente?** 2019. Disponível em: <https://fgvprojetos.fgv.br/noticias/o-que-e-uma-cidade-inteligente>. Acesso em: 28 de julho de 2021. Citado na página 1.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado 2 vezes nas páginas 14 e 17.

GéRON, A. **Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems**. Sebastopol, CA: O'Reilly Media, 2017. Citado na página 4.

GöREN, S. et al. On-street parking spot detection for smart cities. p. 292–295, 2019. Citado na página 16.

HARRIS, C. R. et al. Array programming with NumPy. **Nature**, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>. Citado na página 24.

HE, K. et al. Mask r-cnn. In: **2017 IEEE International Conference on Computer Vision (ICCV)**. [S.l.: s.n.], 2017. p. 2980–2988. Citado na página 15.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. Citado na página 24.

KALBANDE, A. **Best Python Libraries For Machine Learning**. 2020. Disponível em: <https://www.fireblazeaischool.in/blogs/python-libraries-for-machine-learning/>. Acesso em: 10 de outubro de 2021. Citado na página 23.

KHAN, S. et al. **A Guide to Convolutional Neural Networks for Computer Vision**. [S.l.: s.n.], 2018. Citado 3 vezes nas páginas 12, 13 e 14.

KLUYVER, T. et al. **Jupyter Notebooks – a publishing format for reproducible computational workflows**. 2016. 87 - 90 p. Citado na página 24.

LARANJEIRA, C. **Curso de Redes Neurais Convolucionais: Deep Learning com PyTorch**. 2018. Disponível em: <https://cursos.alura.com.br/course/cnn-redes-neurais-convolucionais-deep-learning-pytorch>. Acesso em: 05 de setembro de 2021. Citado 2 vezes nas páginas 12 e 13.

LECUN, Y.; BENGIO, Y. Convolutional networks for images, speech and time series. In: _____. **The Handbook of Brain Theory and Neural Networks**. [S.l.]: The MIT Press, 1995. p. 255–258. Citado na página 11.

MEDURI, P.; ESTEBANEZ, A. A brief review of convolutional neural networks based solutions for smart parking systems. p. 454–457, 2018. Citado na página 16.

METTUPALLY, S. N. R.; MENON, V. A smart eco-system for parking detection using deep learning and big data analytics. In: **2019 SoutheastCon**. [S.l.: s.n.], 2019. p. 1–4. Citado 2 vezes nas páginas 15 e 37.

MITCHELL, T. M. **Machine Learning**. New York: McGraw-Hill, 1997. ISBN 978-0-07-042807-2. Citado na página 3.

NEEL. **Training Neural Networks with Genetic Algorithms**. 2015. Disponível em: <https://blog.abhranil.net/2015/03/03/training-neural-networks-with-genetic-algorithms/>. Acesso em: 05 de outubro de 2021. Citado 2 vezes nas páginas 5 e 6.

NIELSEN, R. H. Theory of the backpropagation neural network. In: **Proceedings of the International Joint Conference on Neural Networks (Washington, DC)**. [S.l.]: Piscataway, NJ: IEEE, 1989. I, p. 593–605. Citado na página 3.

PALMIERE, S. E. **Rede Perceptron de uma única camada**. 2016. Disponível em: <https://www.embarcados.com.br/rede-perceptron-de-uma-unica-camada/>. Acesso em: 02 de outubro de 2021. Citado na página 3.

PANHAN, A. M. **Cidades Inteligentes: a Tecnologia Como Solução de Problemas Urbanos!** 2020. Disponível em: <https://www.vivadecora.com.br/pro/curiosidades/cidades-inteligentes/>. Acesso em: 28 de julho de 2021. Citado na página 1.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning Representations by Back-propagating Errors. **Nature**, v. 323, n. 6088, p. 533–536, 1986. Disponível em: <http://www.nature.com/articles/323533a0>. Citado na página 9.

SANDLER, M. et al. Mobilenetv2: Inverted residuals and linear bottlenecks. In: **2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 4510–4520. Citado na página 24.

SIECK, N.; CALPIN, C.; ALMALAG, M. Machine vision smart parking using internet of things (iots) in a smart university. p. 1–6, 2020. Citado 3 vezes nas páginas 1, 15 e 37.

SOARES, A. da S. **Algoritmo da retropropagação de erros (Backpropagation) para redes multi-layer perceptron**. 2018. Disponível em: <https://youtu.be/DGNbd2FGw2s>. Acesso em: 05 de outubro de 2021. Citado 4 vezes nas páginas 6, 8, 9 e 10.

TEAM, T. pandas development. **pandas-dev/pandas: Pandas**. Zenodo, 2020. Disponível em: <https://doi.org/10.5281/zenodo.3509134>. Citado na página 24.

Apêndices

parking_crop_Pedro

October 12, 2021

1 Apêndice A

[1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "last_expr"
```

[2]:
```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import xml.etree.ElementTree as ET
import uuid
```

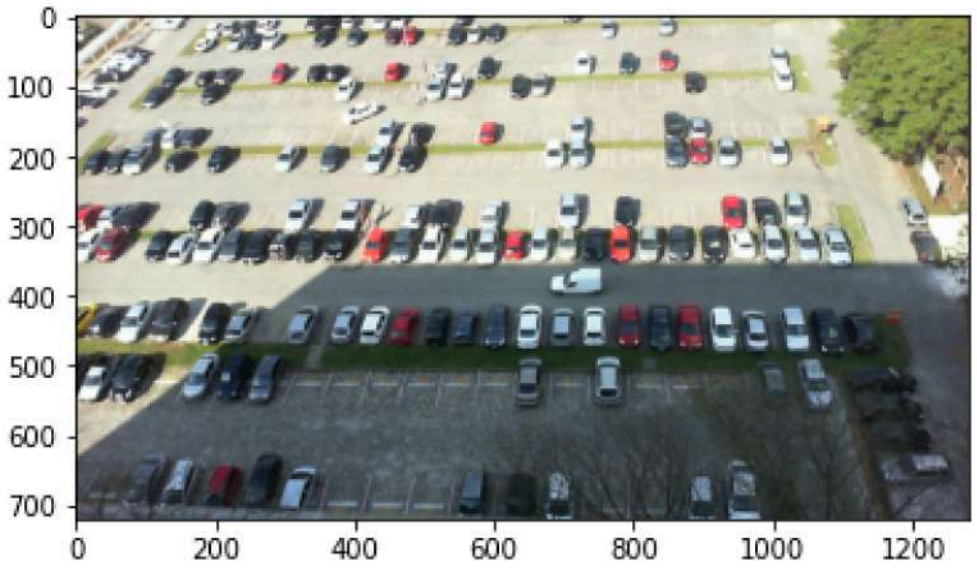
[3]:
```
BASE_PATH = ['../..','datasets']

DATASET_DIR = os.path.join(BASE_PATH[0],BASE_PATH[1],'PKLot')

path = '/PKLot/PUCPR/todas'
vagas = 100
pontos = vagas*4
path = os.path.normpath(path)
path = path.split(os.sep)

IMAGE_DIR = os.path.join(DATASET_DIR,path[2],path[3])
XML_DIR = IMAGE_DIR
SAVE_DIR = os.path.join(DATASET_DIR,path[2])

IMAGE_PATH = []
XML_PATH = []
def criarLista():
    for file in os.listdir(IMAGE_DIR):
        if file.endswith(".jpg"):

            file = os.path.join(IMAGE_DIR,file)
            IMAGE_PATH.append(file)
    IMAGE_PATH.sort()

    for file in os.listdir(IMAGE_DIR):
```

```
        if file.endswith(".xml"):

            file = os.path.join(XML_DIR,file)
            XML_PATH.append(file)
    XML_PATH.sort()


criarLista()
#debug
print(IMAGE_DIR)
print(SAVE_DIR)
print(IMAGE_PATH[:2])
print(XML_PATH[:2])
```

```
../../datasets/PKLot/PUCPR/todas
../../datasets/PKLot/PUCPR
['../../datasets/PKLot/PUCPR/todas/2012-09-11_15_16_58.jpg',
'../../datasets/PKLot/PUCPR/todas/2012-09-11_15_27_08.jpg']
['../../datasets/PKLot/PUCPR/todas/2012-09-11_15_16_58.xml',
'../../datasets/PKLot/PUCPR/todas/2012-09-11_15_27_08.xml']
```

[4]:
```
image = plt.imread(IMAGE_PATH[0])
plt.imshow(image)
```

[4]:
```
<matplotlib.image.AxesImage at 0x7f40aafdaeb0>
```



1.1 Carrega o XML e filtra os pontos X e Y.

[5]:
```
tree = ET.parse(XML_PATH[0])
root = tree.getroot()

pts = np.empty(0,np.int32)
vaga = np.empty(0,np.int32)

for neighbor in root.iter('point'):
    #print(neighbor.attrib)
    x,y = neighbor.attrib.values()
    x = int(x)
    y = int(y)
    a = [x,y]
    pts = np.append(pts,[x,y])

for neighbor in root.iter('space'):
    #print(neighbor.attrib)
    if(len(neighbor.attrib.values())==2):
        _,occupied = neighbor.attrib.values()
        occupied = int(occupied)
    if(len(neighbor.attrib.values())==1):
        occupied=0

    vaga = np.append(vaga,occupied)

#debug
print(pts[0:4])
print(vaga[0:vagas])
```

```
[278 230 290 186]
[1 0 1 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 1 0 1
 0 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 0 1]
```

1.2 Organiza os pares XY de cada Ponto.

[6]:
```
paresXY = np.array(np.zeros((pontos,2)),np.int32)
j=0
for i in range(pontos):
    paresXY[i] = pts[j:j+2]
    j = j+2

#debug
paresXY[0:4]
```

[6]:
```
array([[278, 230],
       [290, 186],
```

```
       [324, 185],
       [308, 230]], dtype=int32)
```

1.3 CROP (Função que segmenta e salva as imagens segmentadas)

[7]:
```
def crop(j=0,i=0,vaga=2,lista=5):
    rect = cv2.boundingRect(paresXY[j:j+4])
    x,y,w,h = rect
    croped = im2[y:y+h, x:x+w].copy()

    pts = paresXY[j:j+4] - paresXY[j:j+4].min(axis=0)
    mask = np.zeros(croped.shape[:2], croped.dtype)
    cv2.drawContours(mask, [pts], -1, (255, 255, 255), -1, cv2.LINE_AA)

    ## (3) do bit-op
    dst = cv2.bitwise_and(croped, croped, mask=mask) #background preto

    bg = np.ones_like(croped, np.uint8)*255
    cv2.bitwise_not(bg,bg, mask=mask)
    dst2 = bg+ dst #background branco

    image = cv2.resize(dst2, (128,128), interpolation = cv2.INTER_CUBIC)
    arquivo = 'vaga{}_ocupado{}_{}.jpg'.format(i,vaga,lista)

    path=os.path.join(SAVE_DIR,'output')
    if not os.path.exists(path):
        os.makedirs(path)

    path = os.path.join(SAVE_DIR,'output','livre')
    if not os.path.exists(path):
        os.makedirs(path)

    path = os.path.join(SAVE_DIR,'output','ocupada')
    if not os.path.exists(path):
        os.makedirs(path)

    if vaga:
        path=os.path.join(SAVE_DIR,'output','ocupada',arquivo)
        cv2.imwrite(path, image)

    else:
        path=os.path.join(SAVE_DIR,'output','livre',arquivo)
        cv2.imwrite(path, image)
```

1.4 Cria as delimitações a partir dos pontos do XML

1.5 Exibe a imagem completa com as segmentações

1.6 Chama a função crop para cada vaga delimitada

1.7 CROP DA LISTA

[9]:
```
%%time

for lista in range (0,len(IMAGE_PATH),6):

    #PARTE 1

    tree = ET.parse(XML_PATH[lista])
    root = tree.getroot()

    pts = np.empty(0,np.int32)
    vaga = np.empty(0,np.int32)

    for neighbor in root.iter('point'):
        x,y = neighbor.attrib.values()
        x = int(x)
        y = int(y)
        a = [x,y]
        pts = np.append(pts,[x,y])

    for neighbor in root.iter('space'):
        if(len(neighbor.attrib.values())==2):
            _,occupied = neighbor.attrib.values()
            occupied = int(occupied)
        if(len(neighbor.attrib.values())==1):
            occupied=0

        vaga = np.append(vaga,occupied)


    #PARTE 2


    paresXY = np.array(np.zeros((pontos,2)),np.int32)
    j=0
    for i in range(pontos):
        paresXY[i] = pts[j:j+2]
        j = j+2

    #PARTE 3
```

```
    im = cv2.imread(IMAGE_PATH[lista])
    im2 = im.copy()
    if im is None:
        sys.exit("A imagem não foi carregada.")

    j=0
    for i in range(len(paresXY)//4):
        cv2.polylines(im,[paresXY[j:j+4]],True,(0,255,255),2)
        crop(j,i,vaga[i],lista)
        j=j+4
```

```
CPU times: user 57.6 s, sys: 3.36 s, total: 1min
Wall time: 1min 1s
```

parking_Pedro_v2.1

October 12, 2021

1 Apêndice B

[1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "last_expr"
```

1.1 IMPORTAR BIBLIOTECAS

[29]:
```
from enum import Enum
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
import sys
import PIL
import PIL.Image
import xml.etree.ElementTree as ET
import datetime
import tensorflow as tf
from tensorboard.plugins.hparams import api as hp
from tensorflow import keras

gpu = tf.test.gpu_device_name()
if gpu == '':
    os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
```

1.2 TREINAR/CARREGAR

[3]:
```
flag = 1 #treinar modelo
flag = 0 #carregar modelo !! (Comente para treinar e salvar)
```

1.3 CONFIGURAÇÕES.

[4]:
```
input_shape = (160,160,3)
batch = 32
```

1.3.1 Importar dataset.

[5]:
```
BASE_PATH = ['../..','datasets']
DATASETS_DIR=['PKLot','UFPR04','UFPR05','PUCPR']

class DATASETS(Enum):
    UFPR04 = os.path.
 ↪join(BASE_PATH[0],BASE_PATH[1],DATASETS_DIR[0],DATASETS_DIR[1])
    UFPR05 = os.path.
 ↪join(BASE_PATH[0],BASE_PATH[1],DATASETS_DIR[0],DATASETS_DIR[2])
    PUCPR  = os.path.
 ↪join(BASE_PATH[0],BASE_PATH[1],DATASETS_DIR[0],DATASETS_DIR[3])
```

[6]:
```
DATASET_DIR = DATASETS.UFPR05.value

if DATASET_DIR == DATASETS.UFPR04.value:
    vagas  = 28
    pontos = vagas*4
    nVagas = 28
    nVagas = vagas - nVagas


if DATASET_DIR == DATASETS.UFPR05.value:
    vagas  = 40
    pontos = vagas*4
    pontos = vagas*4
    nVagas = 37
    nVagas = vagas - nVagas

if DATASET_DIR == DATASETS.PUCPR.value:
    vagas  = 100
    pontos = vagas*4
    nVagas = 100
    nVagas = vagas - nVagas
```

1.3.2 Configurar dataset

[7]:
```
IMAGE_DIR = os.path.join(DATASET_DIR,'todas')
XML_DIR = IMAGE_DIR


IMAGE_LIST = []
XML_LIST = []
def criarLista():
    for file in os.listdir(IMAGE_DIR):
        if file.endswith(".jpg"):
```

```
            file = os.path.join(IMAGE_DIR,file)
            IMAGE_LIST.append(file)
    IMAGE_LIST.sort()

    for file in os.listdir(IMAGE_DIR):
        if file.endswith(".xml"):

            file = os.path.join(XML_DIR,file)
            XML_LIST.append(file)
    XML_LIST.sort()


criarLista()
#debug
print(IMAGE_DIR)
print(XML_DIR)
print(IMAGE_LIST[:2])
print(XML_LIST[:2])
```

```
../../datasets/PKLot/UFPR05/todas
../../datasets/PKLot/UFPR05/todas
['../../datasets/PKLot/UFPR05/todas/2013-02-22_06_05_00.jpg',
'../../datasets/PKLot/UFPR05/todas/2013-02-22_06_10_00.jpg']
['../../datasets/PKLot/UFPR05/todas/2013-02-22_06_05_00.xml',
'../../datasets/PKLot/UFPR05/todas/2013-02-22_06_10_00.xml']
```

1.4 CARREGAR ENTRADAS E SPLIT DA BASE.

[8]:
```
DADOS_DIR = os.path.join(DATASET_DIR,'output')
X = tf.keras.preprocessing.image_dataset_from_directory(DADOS_DIR,␣
 ↪image_size=input_shape[:2], batch_size=batch, label_mode='binary',
                                                    shuffle=True, seed=123,␣
 ↪color_mode='rgb', validation_split=0.7, subset = 'training')


X_val = tf.keras.preprocessing.image_dataset_from_directory(DADOS_DIR,␣
 ↪image_size=input_shape[:2], batch_size=batch, label_mode='binary',
                                                    shuffle=True, seed=123,␣
 ↪color_mode='rgb', validation_split=0.7, subset = 'validation')

class_names = X.class_names
print('\nClasses: {} em {}'.format(X.class_names,DADOS_DIR))
```

```
Found 27680 files belonging to 2 classes.
Using 8304 files for training.
Found 27680 files belonging to 2 classes.
Using 19376 files for validation.
```

```
Classes: ['livre', 'ocupada'] em ../../datasets/PKLot/UFPR05/output
```

1.4.1 Split - devel e teste

[9]:
```
val_batches = tf.data.experimental.cardinality(X_val)
X_test = X_val.take(val_batches // 5)
X_val  = X_val.skip(val_batches // 5)

print('Lotes de validação: %d' % tf.data.experimental.cardinality(X_val))
print('Lotes de Teste: %d' % tf.data.experimental.cardinality(X_test))
```

```
Lotes de validação: 485
Lotes de Teste: 121
```

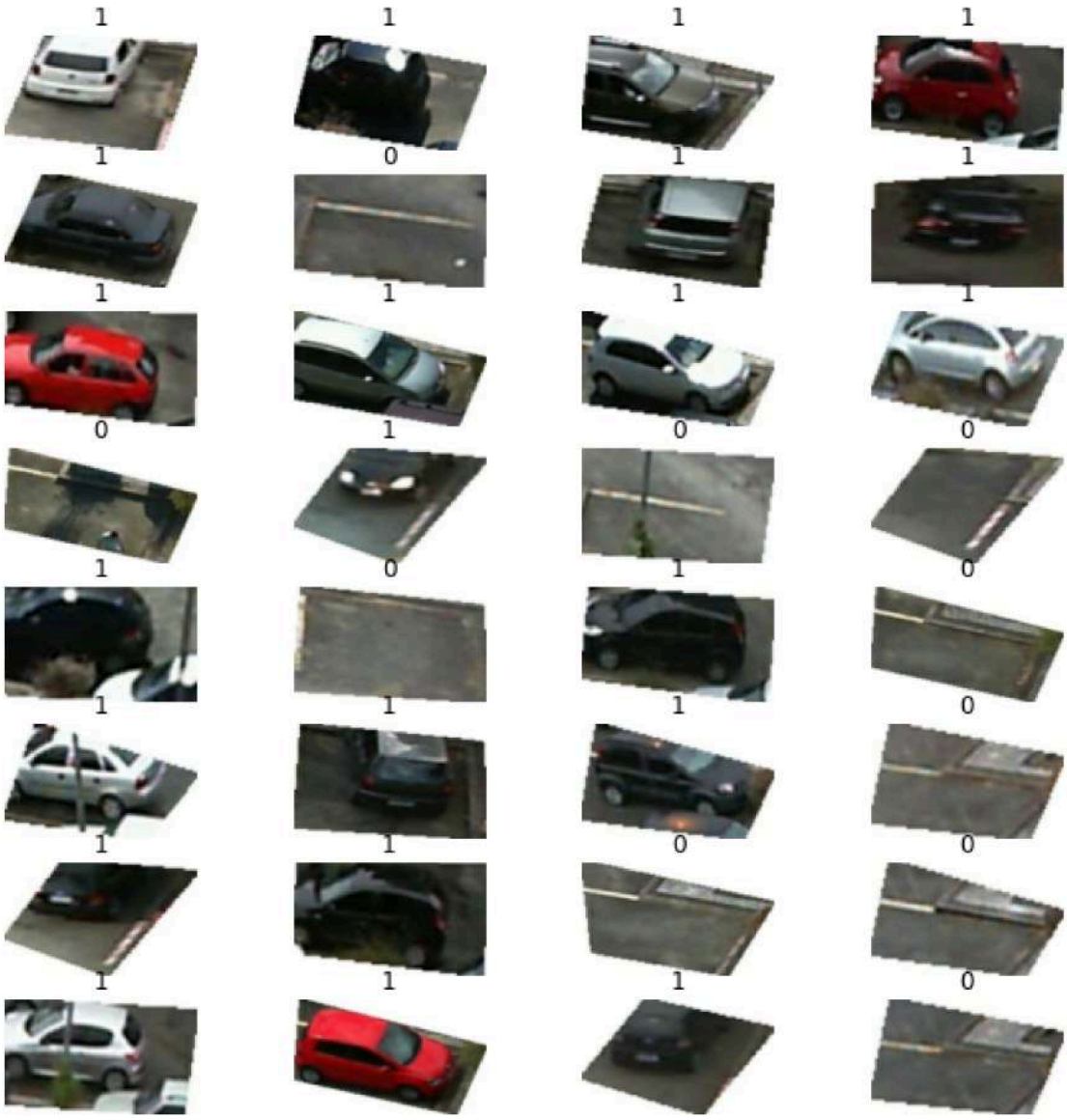
1.4.2 Otimização do pipeline

[10]:
```
AUTOTUNE = tf.data.AUTOTUNE

X = X.prefetch(buffer_size=AUTOTUNE)
X_val = X_val.prefetch(buffer_size=AUTOTUNE)
X_test = X_test.prefetch(buffer_size=AUTOTUNE)
```

1.4.3 Testando o supervisor.

[11]:
```
plt.figure(figsize=(10, 10))
for images, labels in X_test.take(1):
    for i in range(32):
        ax = plt.subplot(8, 4, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"),aspect=0.6,␣
 ↪extent=[0,100,0,100])
        plt.title(int(labels[i]))
        plt.axis("off")
```

```
2021-10-12 00:01:37.333627: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)
```



1.5 EXEMPLO DE IMAGEM NÃO SEGMENTADA.

[12]:
```
sort = np.random.randint(len(IMAGE_LIST))

PIL.Image.open(IMAGE_LIST[sort])
```

[12]:

1.6 TRANSFER LEARNING

1.6.1 Normalizar as imagens para o padrão do modelo da mobilenet_v2

[13]:
```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
rescale = tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5,␣
 ↪offset= -1)
```

1.6.2 Criar o modelo base.

[14]:
```
base_model = tf.keras.applications.MobileNetV2(input_shape=input_shape,
                                                include_top=False,
                                                weights='imagenet')

base_model.trainable = False
#base_model.summary()
```

1.7 MODELO.

[15]:
```
inputs = tf.keras.Input(shape=(160, 160, 3))
x = tf.keras.layers.experimental.preprocessing.Rescaling(1./127.5, offset=␣
 ↪-1)(inputs)
x = base_model(x, training=False) # O modelo contem uma camada de Normalização.␣
 ↪Não destruir os pesos aprendidos
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = tf.keras.layers.Dense(1)(x)
```

```
model = tf.keras.Model(inputs, outputs)

base_learning_rate = 0.0001

model.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 160, 160, 3)]     0
_________________________________________________________________
rescaling_1 (Rescaling)      (None, 160, 160, 3)       0
_________________________________________________________________
mobilenetv2_1.00_160 (Functi (None, 5, 5, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dropout (Dropout)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 1281
=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
```

1.7.1 Configurando o tensorboard

[16]:
```
LOG_DIR = os.path.join(DATASET_DIR,'model','logs','fit')
LOG_DIR2 = os.path.join(LOG_DIR,datetime.datetime.now().
 ↪strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=LOG_DIR2,␣
 ↪histogram_freq=1)
```

```
2021-10-12 00:01:39.848425: I
tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session
initializing.
2021-10-12 00:01:39.848449: I
tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2021-10-12 00:01:39.848811: I
tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear
```

down.

1.7.2 Configurando callback de checkpoint

[17]:
```
CHECKPOINT_DIR = os.path.join(DATASET_DIR,'model')
CHECKPOINT_PATH = CHECKPOINT_DIR+os.sep+'checkpoint'

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=CHECKPOINT_PATH,
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    verbose=1,
    save_best_only=True)
```

1.7.3 Treinar/Carregar.

[18]:
```
if flag:

    history = model.fit(X, epochs=5,␣
 ↪validation_data=X_val,callbacks=[tensorboard_callback, checkpoint_callback])


else:

    LOAD_DIR = os.path.join(DATASET_DIR,'model')

    model = keras.models.load_model(LOAD_DIR)
    #model.load_weights(CHECKPOINT_PATH)
```

1.7.4 Métricas

[19]:
```
if flag:
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

    plt.figure(figsize=(8, 8))
    plt.subplot(2, 1, 1)
    plt.plot(acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.ylabel('Accuracy')
    plt.ylim([min(plt.ylim()),1])
    plt.title('Training and Validation Accuracy')
```

```
    plt.subplot(2, 1, 2)
    plt.plot(loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.ylabel('Cross Entropy')
    plt.ylim([0,0.1])
    plt.title('Training and Validation Loss')
    plt.xlabel('epoch')
    plt.show()

    %load_ext tensorboard
    file_writer = tf.summary.create_file_writer(LOG_DIR2)
    %tensorboard --logdir {LOG_DIR}
```

1.7.5 Salvar modelo

[20]:
```
if flag:
    SAVE_DIR = os.path.split(DATASET_DIR)[0] # Separa (head,tail) -> Indice [0]␣
 ↪= head
    SAVE_DIR = os.path.join(DATASET_DIR,'model')
    model.save(
        SAVE_DIR,
        overwrite=False,
        include_optimizer=True,
        save_format=None,
        signatures=None,
        options=None,
        save_traces=True,
    )
```

1.7.6 Validar modelo na base de teste

[21]:
```
model.load_weights(CHECKPOINT_PATH)
score = model.evaluate(X_test, verbose=2)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
121/121 - 20s - loss: 0.0285 - accuracy: 0.9920
Test loss: 0.02852698042988777
Test accuracy: 0.99199378490448
```

1.7.7 Predição em lote de teste

[22]:
```
image_batch, label_batch = X_test.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
```

```
predictions = tf.nn.sigmoid(predictions)
predictions = tf.where(predictions < 0.5, 0, 1)

for i in range (len(predictions)):
    print('Predição = {}, {} = Label'.format(predictions[i].
 ↪numpy(),label_batch[i].tolist()))

plt.figure(figsize=(10, 10))
for i in range(32):
  ax = plt.subplot(8, 4, i + 1)
  plt.imshow(image_batch[i].astype("uint8"),aspect=0.6, extent=[0,100,0,100])
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predição = 0, [0.0] = Label
Predição = 0, [0.0] = Label
Predição = 1, [1.0] = Label
Predição = 1, [1.0] = Label
Predição = 1, [1.0] = Label
Predição = 1, [1.0] = Label
Predição = 0, [0.0] = Label
Predição = 0, [0.0] = Label
Predição = 1, [1.0] = Label
Predição = 0, [0.0] = Label
Predição = 0, [0.0] = Label
Predição = 1, [1.0] = Label
Predição = 1, [1.0] = Label
Predição = 1, [1.0] = Label
Predição = 0, [0.0] = Label
Predição = 1, [1.0] = Label
Predição = 0, [0.0] = Label
Predição = 1, [1.0] = Label
Predição = 0, [0.0] = Label
Predição = 1, [1.0] = Label
Predição = 1, [1.0] = Label
Predição = 0, [0.0] = Label
Predição = 0, [0.0] = Label
Predição = 0, [0.0] = Label
Predição = 1, [1.0] = Label
Predição = 0, [0.0] = Label
Predição = 0, [0.0] = Label
Predição = 0, [0.0] = Label
Predição = 1, [1.0] = Label
Predição = 0, [0.0] = Label
Predição = 1, [1.0] = Label
Predição = 0, [0.0] = Label
```

1.8 PREDIÇÃO EM TEMPO REAL

1.8.1 Carregar XML e criar os pontos das ROI's.

[23]:
```
tree = ET.parse(XML_LIST[0])
root = tree.getroot()

pts = np.empty(0,np.int32)
vaga = np.empty(0,np.int32)

for neighbor in root.iter('point'):
    x,y = neighbor.attrib.values()
```

```
    x = int(x)
    y = int(y)
    a = [x,y]
    pts = np.append(pts,[x,y])

for neighbor in root.iter('space'):
    if(len(neighbor.attrib.values())==2):
        _,occupied = neighbor.attrib.values()
        occupied = int(occupied)
    if(len(neighbor.attrib.values())==1):
        occupied=0

    vaga = np.append(vaga,occupied)

#debug
print(pts[0:4])
print(vaga[0:vagas])

paresXY = np.array(np.zeros((pontos,2)),np.int32)
j=0
for i in range(pontos):
    paresXY[i] = pts[j:j+2]
    j = j+2

#debug
paresXY[0:4]
```

```
[608 613 741 654]
[0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```

[23]:
```
array([[608, 613],
       [741, 654],
       [775, 582],
       [608, 526]], dtype=int32)
```

1.8.2 Função de segmentação

[24]:
```
def crop(j=0,i=0):
    rect = cv2.boundingRect(paresXY[j:j+4])
    x,y,w,h = rect
    croped = im[y:y+h, x:x+w].copy()

    pts = paresXY[j:j+4] - paresXY[j:j+4].min(axis=0)
    mask = np.zeros(croped.shape[:2], croped.dtype)
    cv2.drawContours(mask, [pts], -1, (255, 255, 255), -1, cv2.LINE_AA)

    ## (3) do bit-op
```

```
    dst = cv2.bitwise_and(croped, croped, mask=mask) #background preto

    bg = np.ones_like(croped, np.uint8)*255
    cv2.bitwise_not(bg,bg, mask=mask)
    dst2 = bg+ dst #background branco

    image = cv2.resize(dst2, input_shape[:2], interpolation = cv2.INTER_CUBIC)

    img = image.copy()
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_array = keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array,  0)

    predictions = model.predict(img_array)

    score = predictions[0]
    score = float(score)
    return score
```

1.8.3 Função para combinar imagem sem predição e imagem sem predição em uma única janela

[25]:
```
def juntar(im2,im,scale=80):

    img = np.hstack((im2, im))

    scale_percent = scale
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
    return resized
```

1.8.4 Carregar e preditar várias imagens aleatórias

[26]:
```
for i in range (2): #definir o número de imagens que será chamado.

    sort = np.random.randint(len(IMAGE_LIST))
    im = cv2.imread(IMAGE_LIST[sort])
    im2 = im.copy()
    if im is None:
        sys.exit("A imagem não foi carregada.")

    j=0
    contador = 0
    for i in range(nVagas,len(paresXY)//4):
```

```
        score = crop(j,i)
        #print(score)
        if((score)<0):
            cv2.polylines(im,[paresXY[j:j+4]],True,(0,255,255),2)
            contador = contador +1
            #print(score)
        j=j+4



    cv2.putText(im,str(contador),(10,700), cv2.FONT_HERSHEY_SIMPLEX,␣
↪4,(255,255,255),5,cv2.LINE_AA)
    img_junta = juntar(im2,im,70)

    cv2.imshow(IMAGE_LIST[sort], img_junta)
    k = cv2.waitKey(0)
    cv2.destroyAllWindows()
```

1.8.5 Carregar Video.

[27]:
```
VIDEO_FILE = os.path.join(DATASET_DIR,'pklot.mp4')
cap = cv2.VideoCapture(VIDEO_FILE)

if cap.isOpened() == False:
    print('Video não encontrado.')

while(cap.isOpened()):
    ret, im = cap.read()
    height, width, _ = im.shape

    if ret == True:

        j=0
        contador = 0


        for i in range(nVagas,len(paresXY)//4):

            score = crop(j,i)
            if((score)<0.7):
                cv2.polylines(im,[paresXY[j:j+4]],True,(0,255,255),2)
                contador = contador +1
            j=j+4

        cv2.putText(im,str(contador),(10,700), cv2.FONT_HERSHEY_SIMPLEX,␣
↪4,(255,255,255),5,cv2.LINE_AA)
```

```
        cv2.imshow('Estacionamento', im)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    else:
        break
cap.release()
cv2.destroyAllWindows()
```

1.9 Debug

[28]:
```
im = plt.imread(IMAGE_LIST[450])
j=0
contador = 0
for i in range(nVagas,len(paresXY)//4):

    score = crop(j,i)
    #print(score)
    if((score)<0.0):
        cv2.polylines(im,[paresXY[j:j+4]],True,(255,255,0),2)
        contador = contador +1
        #print(score)
    j=j+4

fig = plt.figure(figsize=(30, 30))
plt.imshow(im)
plt.title(IMAGE_LIST[350])
plt.axis("off")
```

[28]:
```
(-0.5, 1279.5, 719.5, -0.5)
```


../../datasets/PKLot/UFPR05/todas/2013-02-24_09_45_03.jpg