UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA CIVIL E AMBIENTAL

DETECÇÃO E SEGMENTAÇÃO DE EDIFICAÇÕES UTILIZANDO A ARQUITETURA MASK R-CNN NA CONFECÇÃO DE MAPAS DE USO E COBERTURA DO SOLO

ANDRÉ ESTEVAM COSTA OLIVEIRA

ORIENTADOR: WAGNER DOS SANTOS ALMEIDA

MONOGRAFIA DE PROJETO FINAL EM ENGENHARIA AMBIENTAL II

BRASÍLIA/DF: ABRIL/2022

RESUMO

Acompanhando o crescimento populacional e o desenvolvimento humano as dinâmicas da relação entre o homem e seu meio se tornam cada vez mais complexas, exigindo técnicas e metodologias capazes de reproduzir a realidade fiel e eficientemente. Mapas de Uso e Cobertura do Solo são importantes recursos para instrumentos de manutenção do espaço físico, não só sobre a perspectiva ambiental, mas também de forma pública e social, englobando desde o saneamento básico à vulnerabilidade ambiental por meio de pesquisas, políticas públicas e levantamentos técnicos. O Sensoriamento Remoto, campo no qual se aborda tal assunto, conta com a evolução de sensores de imageamento, dispositivos de localização, transmissores de longo alcance entre outros instrumentos, assim como a rápida escalada do poder de processamento de computadores, permitindo a integração e desenvolvimento de novas tecnologias, as quais também permeiam outros campos do conhecimento como o da Visão Computacional. O campo da Visão Computacional se dedica a estruturação de sistemas visuais artificiais capazes de interpretar e compor entendimentos complexos a respeito do mundo real, demonstrando resultados promissores principalmente devido à área do Aprendizado de Máquina. Intrinsecamente relacionados, os campos da Visão Computacional e Aprendizado de Máquina vêm ganhando bastante atenção devido ao sucesso relativamente recente em tarefas complexas, como a detecção de objetos em imagens, tarefa a qual a arquitetura Mask R-CNN demonstra resultados promissores. Portanto, buscando o refinamento da confecção de Mapas de Uso e Cobertura do Solo, este trabalho busca o desenvolvimento de um modelo de arquitetura Mask R-CNN para a detecção de edificações em imagens, por fim integrando-o com um ambiente computacional de Sistemas de Informação Geográfica.

Palavras-chave: mapa de uso e cobertura do solo, edificação, sensoriamento remoto, aprendizado de máquina, Mask R-CNN, segmentação de instâncias.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA CIVIL E AMBIENTAL

DETECÇÃO E SEGMENTAÇÃO DE EDIFICAÇÕES UTILIZANDO A ARQUITETURA MASK R-CNN NA CONFECÇÃO DE MAPAS DE USO E COBERTURA DO SOLO

ANDRÉ ESTEVAM COSTA OLIVEIRA

MONOGRAFIA DE PROJETO FINAL SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA CIVIL E AMBIENTAL DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM ENGENHARIA AMBIENTAL.

APROVADA POR:

_____

WAGNER SANTOS DE ALMEIDA, DSc (ENC/UNB)
(ORIENTADOR)

_____

NEWTON MOREIRA DE SOUZA, DSc (UnB)
(EXAMINADOR INTERNO)

_____

LENILDO SANTOS DA SILVA, DSc (UnB)
(EXAMINADOR INTERNO)

DATA: BRASÍLIA/DF, 02 DE MAIO DE 2022.

FICHA CATALOGRÁFICA

OLIVEIRA, ANDRÉ ESTEVAM COSTA

Detecção e segmentação de edificações utilizando a arquitetura Mask R-CNN na confecção de mapas de uso e cobertura do solo.

ix, 47 p., 297 mm (ENC/FT/UnB, Bacharel, Engenharia Ambiental, 2022)
Monografia de Projeto Final – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Civil e Ambiental.

1. Classificação 2. Edificação
3. Aprendizado Profundo 4. Mask R-CNN
I. ENC/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

Oliveira, A.E.C. 2022. Detecção e segmentação de edificações utilizando a arquitetura Mask R-CNN na confecção de mapas de uso e cobertura do solo. Monografia de Projeto Final, Departamento de Engenharia Civil e Ambiental, Universidade de Brasília, Brasília, DF, 47p.

CESSÃO DE DIREITOS

NOME DO AUTOR: André Estevam Costa Oliveira

TÍTULO DA MONOGRAFIA DE PROJETO FINAL: Detecção e segmentação de edificações utilizando a arquitetura Mask R-CNN na confecção de mapas de uso e cobertura do solo.

GRAU / ANO: Bacharel em Engenharia Ambiental / 2022

_____

André Estevam Costa Oliveira
andre.estevam.unb@gmail.com

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

ML	*Machine Learning*
DL	*Deep Learning*
CV	*Computer Vision*
SR	Sensoriamento Remoto
VHR	*Very High Resolution*
OD	*Object Detection*
MLP	Perceptron Multicamadas
RNA	Rede Neural Artificial
CNN	*Convolutional Neural Network*
R-CNN	*Region-based Convolucional Neural Networks*
ReLU	*Rectified Linear Unit*
FC	*Fully Connected*
RoI	*Region of Interest*
GPU	*Graphics processing unit*

1. INTRODUÇÃO

Brasília possui seu Plano Diretor de Ordenamento Territorial (SEDUH, 2021) e vem sofrendo um crescimento urbano periférico desordenado e segregador, decorrente de uma explosão demográfica. Em consonância ainda acontece a grilagem de terras públicas e o parcelamento de grandes propriedades privadas em condomínios habitacionais, frequentemente irregulares, intensificando a disparidade da qualidade de vida de classes sociais além do processo de degradação do meio ambiente (Mesquita *et al*., 2017).

Não obstante, as agências reguladoras dispõem de normas e planos a respeito do Uso e Cobertura do Solo, cuja importância está no cumprimento da legislação ambiental, estabelecendo responsabilidades essenciais na manutenção de recursos naturais e saneamento básico. As ações antrópicas como perda ou contaminação de corpos hídricos, habitações insalubres e supressão de mata nativa são exemplos de consequências do descumprimento de normas ou mal planejamento territorial, o que ressalta a importância de políticas públicas sustentadas por estudos do território bem fundamentados.

Diante deste contexto e no que tange o ordenamento territorial e, consequentemente, o desenvolvimento sustentável, faz-se necessária a presença de agentes públicos dotados de ferramentas capazes de acompanhar eficaz e eficientemente a rápida expansão de assentamentos populacionais, além da constituição de estudos ambientais e planos territoriais subsidiados por mapas temáticos precisos a respeito do uso e cobertura do solo.

Com a rápida evolução de tecnologias de sensoriamento remoto empregadas para fim de ordenamento territorial, como bancos de dados em nuvem, veículos não tripulados e sensores de alta resolução, faz-se cada vez mais necessário o uso de técnicas de análise de imagens digitais com altas resoluções espacial e espectral.

No que tange à análise de imagens digitais, o campo de Aprendizado Profundo (DL) e o da Visão Computacional (CV) vem demonstrando ótimos resultados em tarefas como detecção de objetos, classificação temática de imagens e segmentação de instâncias, através da estruturação de sistemas visuais artificiais capazes de interpretar e compor entendimentos complexos a respeito da ocupação do solo (Burger e Burge, 2021).

Em meio à emergência de sistemas complexos no campo da Visão Computacional e considerando os problemas de ordenamento territorial citados, tem-se a oportunidade de dispor de tais tecnologias no âmbito da gestão pública para um processo mais eficiente e eficaz de monitoramento e fiscalização, especificamente na detecção de ocupação antrópica em imagens digitais no acompanhamento de rápidas instaurações de assentamentos populacionais. Tal

problemática é acompanhada por uma ampla gama de metodologias abordando operações de classificação e segmentação semântica. Por sua vez, a implementação de um sistema capaz de operar a segmentação de instâncias, isto é, classificação em nível de pixel tratando cada objeto como uma entidade independente, representa uma evolução na precisão e qualidade do monitoramento.

Os esforços de desenvolvimento e implementação de tais sistemas se concentram em imagens naturais, geralmente cenários urbanos contendo objetos genéricos como bicicleta, pessoa, carro entre outros., sendo sua aplicação no meio do Sensoriamento Remoto (SR) relativamente recente e, quase que, exclusivo a imagens de altíssima resolução espacial (VHR) (Mohanty *et al.*, 2020), limitando a aplicabilidade de tais tecnologias conforme o orçamento disponível. Não obstante, seu potencial é visível através de avanços em técnicas de Super Resolução (Courtrai, Pham e Lefèvre, 2020), reconstrução 3D (Li, S. *et al.*, 2019) e, especialmente, Detecção de Objetos (OD) (Zhao *et al.*, 2017).

Entre os sistemas recentemente considerados estado-da-arte na tarefa de detecção de objetos (OD) em imagens, as arquiteturas que utilizam a abordagem *Region-based Convolutional Neural Network* (R-CNN) apresentam alguns dos melhores desempenhos (Cheng e Han, 2016). Diferentemente de metodologias anteriores as quais, em sua maioria, abordavam localização como um problema de regressão, a R-CNN utiliza do paradigma de proposta de regiões (Girshick *et al.*, 2013a), ou de duas etapas. Tal abordagem mostrou-se eficiente e trabalhos posteriores propuseram diversos aperfeiçoamentos à arquitetura, culminando na arquitetura Mask R-CNN, proposto por He et al. (2017), a qual localiza e segmenta os objetos alvos.

A arquitetura Mask R-CNN emprega múltiplos instrumentos do Aprendizagem de Máquina e de sua subárea, o Aprendizado Profundo. Em sua estrutura básica estão presentes Redes Neurais Artificiais (RNA) e Redes Neurais Convolucionais (CNN) (Aggarwal, 2018). Tais algoritmos são reconhecidos pela sua excelência em tarefas complexas e a arquitetura em questão usufrui destas capacidades no âmbito da Visão Computacional e, mais recentemente, no Sensoriamento Remoto. Entretanto, outra característica nata do campo é sua exigência por vastas quantias de dados, para modelos supervisionados, anotados, motivo o qual explica a apenas recente ascensão do campo. Não obstante, a tarefa de Detecção de Objetos (OD) em imagens aéreas ou orbitais atrai interesse tanto de setores públicos como privados, portanto repositórios públicos de dados são relativamente bem difundidos (Li, K. *et al.*, 2019). Além disto dados de imageamento do território brasileiro é facilmente obtido por meio de imagens de satélite disponibilizadas pelo INPE, algumas com até 1 metro de resolução espacial em ótimos

intervalos de tempo. Somando a flexibilidade de ambientes SIG para manipulação de dados georreferenciados, tem-se cenário fértil para a exploração de técnicas robustas de Aprendizado de Máquina (ML) como ferramenta de suporte na elaboração de mapas temáticos de uso e cobertura do solo.

2. OBJETIVO

2.1 OBJETIVO GERAL

Este trabalho tem por objetivo geral desenvolver e avaliar um modelo de Aprendizado de Máquina, baseado na arquitetura Mask R-CNN, para a detecção de ocupações antrópicas em imagens de Sensoriamento Remoto utilizadas na confecção de mapas de Uso e Cobertura do Solo, disponibilizando sua implementação por meio de uma interface amigável e permitindo integração com ambientes computacionais de SIG.

2.2 OBJETIVOS ESPECÍFICOS

- Extrair dos dados disponíveis das imagens de sensores remotos de alta resolução, conjuntos de amostras em formato condizente com o exigido para o treinamento, validação e teste do modelo.
- Desenvolver o código-fonte do modelo a ser implementado, além do roteiro de treinamento, validação e teste, para a classificação e segmentação via interface amigável.
- Iterativamente treinar e avaliar o modelo de forma a definir sua estrutura ótima, seguido do teste final de desempenho.
- Executar técnicas de classificação de imagem pixel por pixel e por região utilizando métodos clássicos para comparação com o desempenho do modelo desenvolvido nas etapas anteriores.
- Desenvolver interface amigável integrada a um ambiente SIG para fácil execução do modelo e aplicação em mapas temáticos do uso e cobertura do solo.

3. FUNDAMENTAÇÃO TEÓRICA E REVISÃO BIBLIOGRÁFICA

O fluxograma exposto pela Figura 1 apresenta os conceitos aqui abordados para o entendimento do sistema a ser implementado no desenvolvimento de mapas de uso e cobertura do solo.

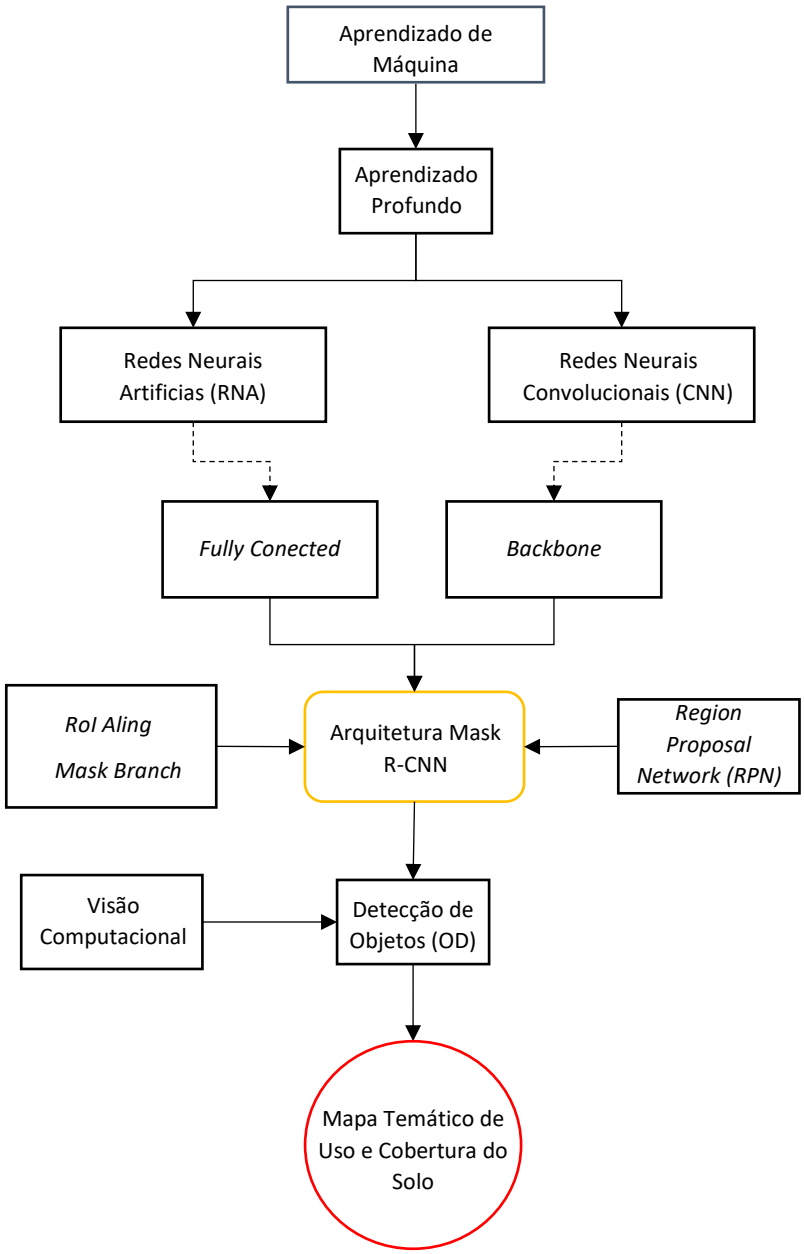


Figura 1 - Fluxograma da Fundamentação Teórica

3.1 CLASSIFICAÇÃO TEMÁTICA, SEGMENTAÇÃO E RECONHECIMENTO

Segundo Crosta (1999), a classificação temática, no que tange ao Sensoriamento Remoto, refere-se à tarefa de identificação e distinção entre determinadas classes temáticas de uso e cobertura do solo, de forma a associar cada pixel da imagem com essas classes, resultando em um mapa temático. A classificação tipo pixel a pixel se dá através da análise do comportamento de suas curvas de reflectância em determinados intervalos de frequência da radiação eletromagnética, intitulados de bandas espectrais, interpretando-os como vetores em um espaço *n dimensional* no intuito de se localizar um hiperplano capaz de delimitar as regiões referentes a cada classe. Este processo de distinção pode ser auxiliado por meio de treinamento das amostras de pixels feita pelo usuário, distinguindo a classificação supervisionada da não-supervisionada.

Segundo Richards e Jia (2006), o procedimento de classificação supervisionada se dá em três etapas: amostragem, treinamento e generalização. Supondo as categorias água, solo e vegetação, a amostragem refere-se à coleta de amostras de pixels referente às três classes mencionadas, o treinamento à fase de delimitação de cada região dentro do espaço *n dimensional* referente às classes utilizando ferramentas estatísticas, e, finalmente, a classificação de cada pixel de acordo com suas propriedades espectrais. A classificação não-supervisionada carece da etapa de amostragem, em que o próprio algoritmo define os melhores agrupamentos de pixels para a etapa de treinamento. Alguns dos classificadores mais utilizados envolvem Máxima Verossimilhança, Distância Mínima e Método do Paralelepípedo.

Entretanto, tal abordagem possui limitações, no sentido de que os classificadores até aqui discutidos utilizam unicamente a informação espectral de cada pixel para abordar o problema, ignorando importante informação das relações entre si, e de suas vizinhanças. De forma a contornar esta deficiência, determinados algoritmos utilizam a detecção de limites entre agrupamentos de pixels "semelhantes", antes da classificação temática em si, recebendo a nomenclatura de "classificadores por região" em contraste aos "classificadores pixel a pixel" (Druck *et al.*, 2014). Para a captura destes limites dá-se o título de segmentação.

Também conhecida como segmentação semântica, esta consiste na designação de regiões de pixels que dividem certa similaridade. No campo da estatística o problema possui a nomenclatura de "análise de clusters" (Bishop, 2011). Na prática a segmentação é abordada por diversos campos compondo dezenas de algoritmos (Szeliski, 2021); entretanto, no campo do Sensoriamento Remoto, tem-se três principais métodos de segmentação: crescimento de regiões, detecção de bordas e detecção de bacias (Druck *et al.*, 2014).

As metodologias abordadas até aqui são amplamente difundidas em aplicações do Sensoriamento Remoto, possuindo papel fundamental em análises com dimensão espacial, além do auxílio às tomadas de decisão (Druck *et al.*,2014). Não obstante, a excelência de tais aplicações concentra-se em problemas geralmente mais abrangentes, nos quais dispensa-se precisão em nível de centímetros ou até metros. Neste sentido, o problema do reconhecimento constitui um maior desafio.

De acordo com Szeliski (2021), reconhecimento, diferentemente de classificação e segmentação, não se ocupa da classificação de cada pixel da imagem de acordo com pré-determinadas categorias mas, sim, da identificação e localização de potenciais alvos dentro da imagem. A princípio ambas as definições são similares; entretanto, a localização supõe uma informação mais precisa a respeito de menores regiões de interesse. Como exemplo, supondo uma malha urbana, enquanto a classificação, ou segmentação semântica, objetivaria a classificação de todos os pixels referentes a tal malha como Urbano e o resto como Não Urbano, o reconhecimento objetivaria a detecção e enquadramento de cada edificação inserida na malha (tarefa conhecida como Detecção de Objetos). Dentre os esforços para a aplicação de tarefas de reconhecimento dentro do escopo do Sensoriamento Remoto algumas abordagens constituem a Análise Orientada a Objetos em Imagens (OBIA ou GEOBIA) e Aprendizado de Máquina (Cheng e Han, 2016).

3.2 APRENDIZADO DE MÁQUINA (ML)

O produto da Ciência da Computação e subcampo da Inteligência Artificial, o Aprendizado de Máquina (ML) pode ser entendido como uma abordagem na concepção de modelos capazes de operar tarefas complexas sem terem sido manualmente programados para tais funções. Muito similar à Estatística, esta capacidade se dá através do ajuste das variáveis do modelo, seus parâmetros, em um processo iterativo chamado “treinamento” (Aggarwal, 2018).

O modelo pode ser classificado entre supervisionado, não supervisionado e semi-supervisionado, o primeiro denotando a necessidade de dados e, adicionalmente, sua correspondente anotação, de acordo com o objetivo final. A “supervisão” constitui-se no processo de comparar, após computação dos dados, a inferência do modelo e o resultado esperado, ajustando seus parâmetros e, consequentemente, seu comportamento à frente de novos cenários. É nesse quesito em que o potencial de algoritmos de ML é alcançado, quando o modelo é capaz de transferir sua cognição firmada no processo de treinamento para dados inexplorados, reconhecido como seu poder de generalização (Skansi, 2018). De forma

simplificada e em termos matemáticos, um modelo de ML pode ser entendido como uma função **y(x)** arbitrariamente grande, a qual recebe o vetor **x** e retorna o vetor $\hat{\boldsymbol{y}}$. A forma precisa da função **y(x)** é definida na fase de treinamento, onde um número suficientemente grande de pares de vetores de entrada **x** e de anotação **y** são entregues ao modelo. A metodologia de ajuste dos parâmetros varia de modelo para modelo, entretanto o processo geralmente envolve a computação da distância entre o inferido $\hat{\boldsymbol{y}}$ e o rotulado **y** por uma função denominada função erro, para uma amostra, e função perda, em se tratando de um conjunto de amostras, as quais procura-se minimizar (Bishop, 2011).

A classe especialmente importante para o entendimento de algoritmos mais robustos como Redes Neurais Artificiais é o *perceptron*, concebido por Frank Rosenblatt na década de 60 (Skansi, 2018). O *perceptron*, em sua forma original, é um modelo de classificação binária, onde o vetor de entrada **x** passa primeiro por uma transformação não linear $\phi(\mathbf{x})$ para então ser utilizado pelo modelo de parâmetro **w** e função de ativação $f(\cdot)$. Sua forma geral é explanada pela Equação (3.1).

$$y(x) = f(\boldsymbol{w}^{\boldsymbol{T}} * \phi(\mathbf{x})) \tag{3.1}$$

A transformação não linear tipicamente traduz-se na adição de um componente de bias *b*, enquanto a função de ativação $f(\cdot)$ traduz-se na função exposta pela Equação (3.1). A partir deste modelo foi proposto diferentes variantes conforme se distingue a função de ativação, transformação não linear e função erro correspondente. O modelo de Regressão Logística pode ser entendido como uma destas variantes. Dotado de similar estrutura, a Regressão Logística utiliza de uma função sigmoidal para inferência de valores entre 0 e 1, ou de probabilidade.

$$f(a) = \begin{cases} +1, a \geq 0 \\ -1, a < 0 \end{cases} (K) \tag{3.2}$$

A utilização de *perceptrons* em problemas de classificação binária pode ser facilmente expandida para problemas de classificação multi-classe, estruturando cada *perceptron* como uma unidade em uma estrutura de camada em que cada unidade é responsável por determinada classe, adequando função de ativação e função erro. A Figura 2 exemplifica 1 unidade de *perceptron* para classificação binária e uma camada de 3 unidades para classificação ternária.

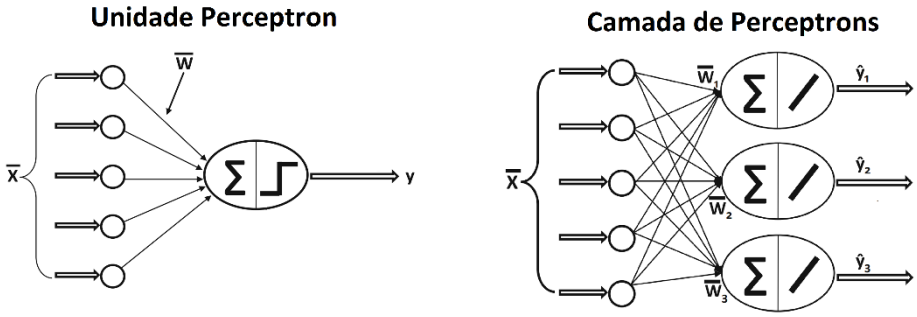


Figura 2 - Exemplo de Unidade *Perceptron* e uma Camada de *Perceptrons*

Fonte: Aggarwal (2018) (modificado)

Poderia se supor que tal lógica pode ser utilizada para expandir além da simples classificação. Agregando camada sobre camada, poderia se ter complexa rede onde cada unidade passa seu "entendimento" sobre as informações de camadas anteriores para camadas mais à frente e assim por diante, em que quão maior a profundidade mais abstrata torna-se a informação transportada. Tal intuição convém a ideia geral de Redes Neurais Artificiais (Aggarwal, 2018).

3.2.1 Redes Neurais Artificiais (RNA)

As Redes Neurais Artificiais foram concebidas de forma a simular o funcionamento básico de neurônios e suas redes de sinapses, primeiro por McCulloch em 1943. O neurônio, unidade básica do sistema nervoso, recebe informação em forma de impulsos elétricos através de seus dendritos, os quais, após determinado processo bioquímico, podem ativar ou não o repasse da informação processada, além de alterar o estado das sinapses. Tal estrutura permite respostas complexas e adaptação de acordo com estímulos exteriores, aonde a "força" das sinapses ditam o comportamento geral da rede. Através destes princípios é possível a formulação matemática de uma rede de unidades interligadas em camadas, cada qual responsável pelo repasse de informação computada a partir da informação recebida pelas unidades anteriores (Skansi, 2018).

Em Redes Neurais Artificiais as unidades de processamento também se intitulam neurônios, ou nós, os quais recebem informação e as pondera com seus pesos, ou parâmetros, espelhando as forças sinápticas do sistema nervoso. Este tipo de rede, na qual a informação é sempre passada adiante, é intitulada *feedforward,* o que significa que a informação sempre é transportada no sentido de camadas iniciais para camadas finais. A Equação (3.3) expõe a forma geral de uma camada.

$$y(\boldsymbol{x}, \boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_j * \phi_j(\boldsymbol{x})\right) \tag{3.3}$$

Onde:

w = vetor peso

x = vetor de entrada

f(·) = função de ativação

ϕ(x) = transformação não linear

j = índice do nó

M = quantidade de nós

A rede Perceptron Multicamada (MLP) é uma rede *feedforward*, dividida entre camada de entrada, camadas intermediárias, também conhecidas como ocultas, e camada de saída, a primeira encarregada de receber os dados e a camada de saída de retornar à solução. As camadas conhecidas como ocultas receberam tal nomenclatura diante do desconhecimento das características intrínsecas da informação repassada e do processamento em si; entretanto, um poderia argumentar que as camadas intermediárias são as de maior importância, as quais definem a maior parte da função e capturam desde relacionamentos simples a complexos dos dados. As operações matemáticas de uma unidade da rede podem, então, ser divididas em duas etapas: ponderação das informações recebidas e ativação, a primeira sendo comum a qualquer unidade da rede e a função de ativação passível de modificação de acordo com a camada em que se encontra. As Equações (3.4) e (3.5) demonstram a decomposição da Equação (3.3) para as duas etapas mencionadas, no caso em que se têm múltiplas camadas compostas por múltiplas unidades.

$$a_j = \sum_{i=1}^{D} w_{ji} x_i + w_{j0} \tag{3.4}$$

Onde:

w = vetor peso

w_0 = vetor bias

a_j = vetor ativação

j = índice do nó

i = índice do nó de entrada

xi = vetor do nó de entrada

D = quantia de nós de entrada

$$z_j = h(a_j) \tag{3.5}$$

Onde:

z_j = vetor de saída

a_j = vetor ativação

h(·) = função de ativação

A função de ativação muda de acordo com a camada, como já mencionado, entretanto é difundido a utilização da função de ativação Unidade Linear Retificada (ReLU) para camadas intermediárias e, para camadas de saída, função sigmoide para soluções limitadas ao intervalo 0 e 1 ou tanh para entre -1 e 1. Para melhor visualização, enquanto a Figura 2 exemplifica bem uma unidade e uma camada, a Figura 3 demonstra um exemplo de uma rede propriamente dita.

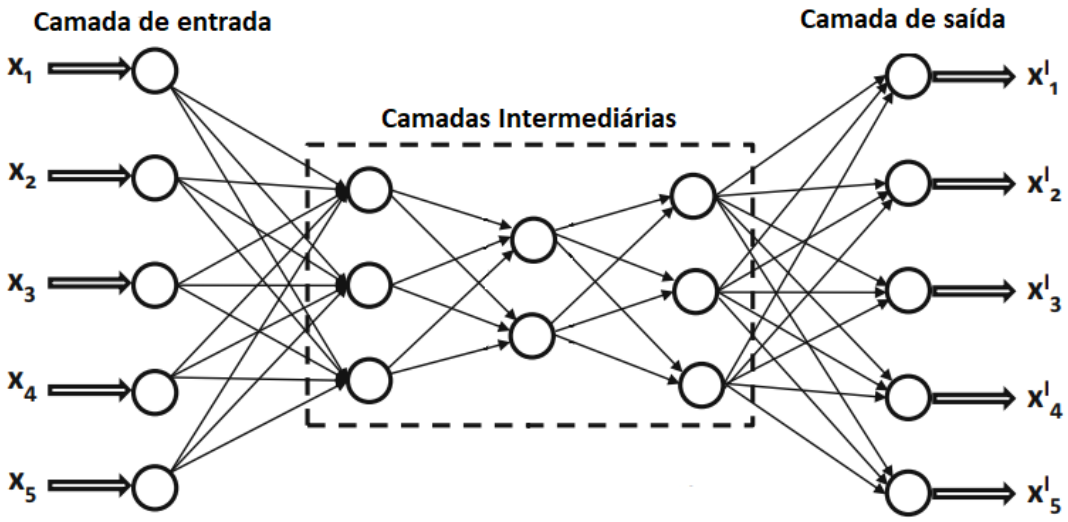


Figura 3 - Rede Neural Artificial genérica

Fonte:Aggarwal (2018) (modificado)

Em contrapartida à unidade do *perceptron*, que era capaz de soluções apenas de problemas lineares, RNAs possuem alta capacidade de solução para problemas não lineares, teoricamente capaz de simular qualquer função matemática, dada devida profundidade e quantia suficiente

de dados para treinamento. Entretanto, e motivo pelo qual em seus primórdios as RNAs não desfrutavam de tanta popularidade como atualmente, é que quão mais profunda é a rede maior a sua demanda por dados para ajuste e mais sujeita a sobreajuste, onde os parâmetros são superajustados aos dados de entrada e o modelo perde seu poder de generalização.

Como já mencionado, o ajuste dos parâmetros do modelo é em função da distância entre o inferido e o anotado (*ground-truth)*, ditada pela função erro a qual toma forma de acordo com a finalidade do modelo. Algumas funções clássicas empregada como a função erro são: Erro Quadrático Médio (MSE), log da Máxima Verossimilhança e Perda de Dobradiça (Bishop, 2011).

Após computação da inferência e cálculo do erro este é "transportado" em sentido contrário até a primeira camada, em um processo chamado propagação de erro. Neste processo, tratando-se de uma função erro diferençável, a ideia é calcular a derivada da função perda, a qual possui o mesmo princípio da função erro, porém através de várias amostras, em relação a cada parâmetro do modelo de forma a modificá-los no sentido do mínimo local da função perda. Em um caso natural tal premissa seria irracional, uma vez que se trata de centenas e, potencialmente, milhares de parâmetros, possivelmente dispostos em complexas funções. Entretanto, para RNAs, devido à disposição e relativa simplicidade da computação de cada nó, pode-se abordar o problema de forma simplificada através de um grafo computacional (Aggarwal, 2018). Para melhor entendimento tal conjectura é exposta na Figura 4 a qual apresenta a diferenciação da função erro ao longo da unidade utilizando as premissas de um grafo computacional.

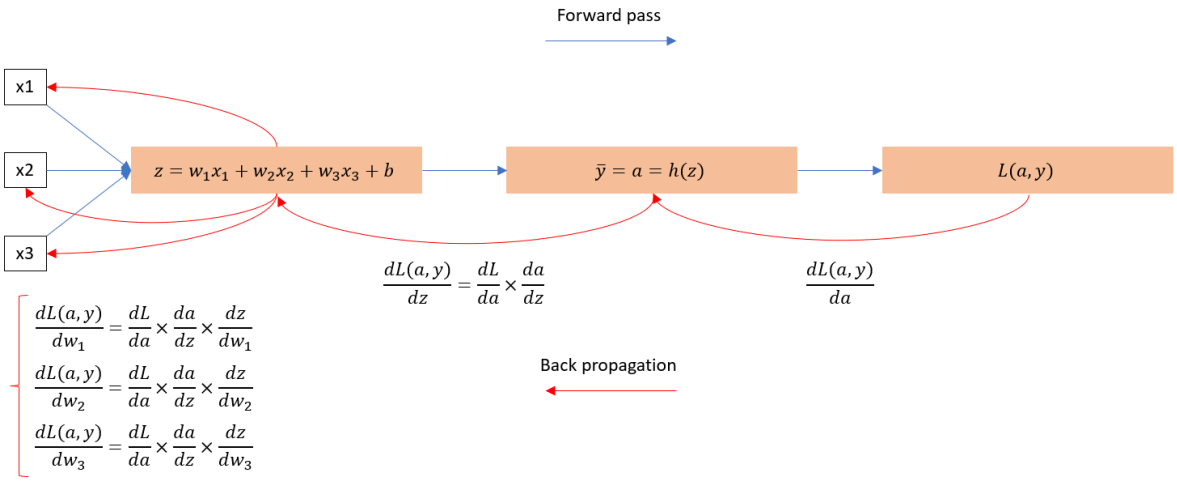


Figura 4 – Grafo computacional de um perceptron

Similarmente à estrutura neurológica animal, os neurônios, além de possuir grande diversidade de tipos entre si, podem formar diferentes estruturas cada qual especializada em determinada finalidade. Diante desta flexibilidade tem-se a concepção de diversas redes encarregadas das mais diversas funções, e assim expandindo sua aplicabilidade nos mais diversos campos. Reconhecimento facial, processamento de linguagem natural, redes robóticas e Detecção de Objetos sendo alguns exemplos (Skansi, 2018).

3.2.2 Redes Neurais Convolucionais (CNN)

As Redes Neurais Convolucionais (CNN) são formas especiais de RNAs, programadas para receber dados matriciais ao invés de vetores. Diversas formas de dados podem ser estruturadas como matrizes, especialmente dados temporais e espaciais, porém a grande maioria das aplicações de CNNs se concentram em imagens. Sua estrutura elementar é o filtro, também conhecido como kernel.

Difundido também nas áreas do Sensoriamento Remoto e de Processamento de Imagens Digitais, os filtros são úteis para diversas finalidades como detecção de limites entre agrupamentos de pixels com similares intensidades de luz refletida (Richards e Jia, 2006) e suavização de imagem (Burger e Burge, 2021), sendo alguns clássicos os filtros de Sobel e filtros de Gauss. Estes são mais conhecidos ao tomarem a forma de uma “janela móvel”, a qual escaneia a matriz original com a finalidade de gerar outra matriz, geralmente de menor dimensão, com pixels representando outro tipo de informação.

Matematicamente, pode-se descrever um filtro como uma função a qual toma o valor de uma região de pixels, computando um único valor em função dos demais, o processo recebendo a nomenclatura de convolução. As dimensões e o comportamento no “deslizamento” de tais kernels são customizáveis e exercem forte influência sobre a matriz resultante. A Figura 5 apresenta um exemplo do procedimento com um filtro aleatório.

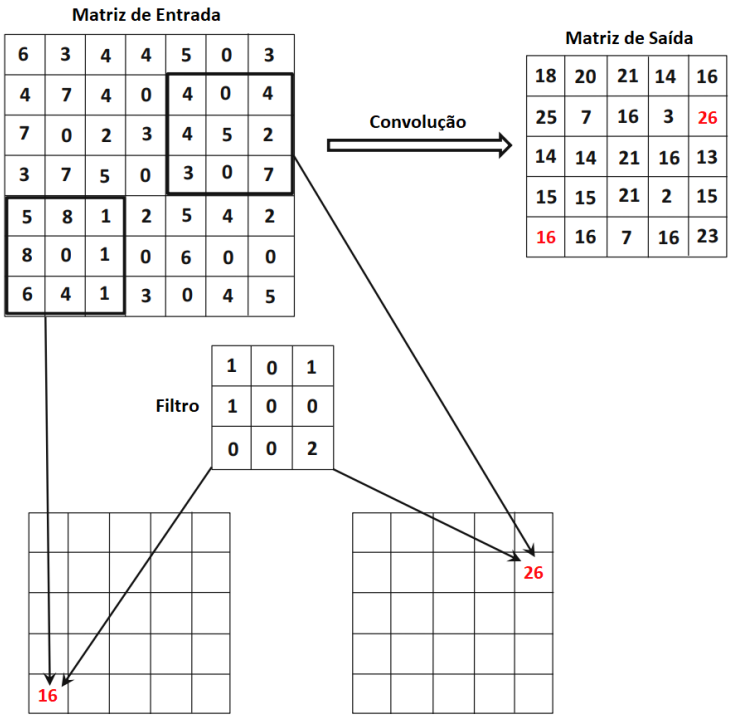


Figura 5 - Convolução com um filtro aleatório

Fonte: Aggarwal (2018) (modificado)

Na utilização deste aparato na composição de sua estrutura geral, CNNs são dotadas de três importantes características: campos receptivos locais, compartilhamento de pesos e subamostragem.

Como discutido, a utilização do filtro resulta em uma matriz cujos elementos foram computados a respeito de determinadas regiões da matriz original, os quais levam o nome de campo receptivo, que nada mais é que a região da matriz de origem a qual o pixel em questão baseou-se para o seu valor. Neste sentido pode-se entender outra característica de CNNs, a denominada "invariância de tradução".

A invariância de tradução remete à capacidade do sistema de captar determinado relacionamento ou padrão nos dados invariavelmente de deslocamentos, neste caso, espaciais. Ou seja, um modelo especializado na detecção de bananas provavelmente será capaz de detectá-las invariavelmente de sua localização na imagem.

As CNNs se aproximam de RNAs no sentido que ambas são estruturadas de forma que a informação é propagada através de uma rede, gradativamente elevando seu nível de complexidade e abstração quão mais profundo se encontra. A forma que tal premissa se dá na utilização de filtros é através da subamostragem. Como já mencionado, a dimensão da matriz é

reduzida ao trespassar o procedimento de convolução, portanto a informação contida em seus elementos é, de uma forma ou outra, agregada. Neste sentido, ao invés do filtro ser manualmente programado para a extração de informações pré-determinadas, como bordas, o objetivo final é que estes sejam capazes de capturar relações e entendimentos muito mais abstratos. Este comportamento é alcançado via estruturação de camadas e canais. Enquanto se tem *n* quantia de nós por camada, CNNs portam *n* quantia de filtros, cada qual responsável por extrair determinada informação da matriz de entrada. Após cada filtro escanear a matriz de entrada, no mesmo formato de "janelas deslizantes", as matrizes resultantes são empilhadas, gerando uma nova matriz de profundidade, ou quantidade de canais, *n*. Assim tem-se matrizes gradativamente menores em relação à altura e comprimento, porém maiores em profundidade, chamadas *feature maps*. Matematicamente, este procedimento significa a expansão do espaço vetorial de cada pixel, na esperança de que, em seu relacionamento com os demais, seja possível a extração do hiperplano para problemas complexos (Aggarwal, 2018).

Invariavelmente da estrutura utilizada, a informação extraída pelos filtros será de acordo com os valores dos elementos do kernel. A forma pela qual se determina tais valores é os parametrizando, tornando seus valores justamente os pesos a serem determinados através de treinamento. Assim, se usufrui das mesmas premissas de RNAs, em que os parâmetros a serem treinados ditarão o comportamento geral da rede. Por final, após computação da matriz resultante, adiciona-se o vetor bias e aplica-se a função de ativação, de forma a aproximá-lo do comportamento de um *perceptron* e fornecer não linearidade à função. Os mesmos mecanismos de treinamento de RNAs são aplicados aqui, através da propagação do erro no sentido contrário, iterativamente ajustando seus parâmetros. A estrutura do procedimento é apresentada na Figura 6, a qual expõe o caso com dois filtros.

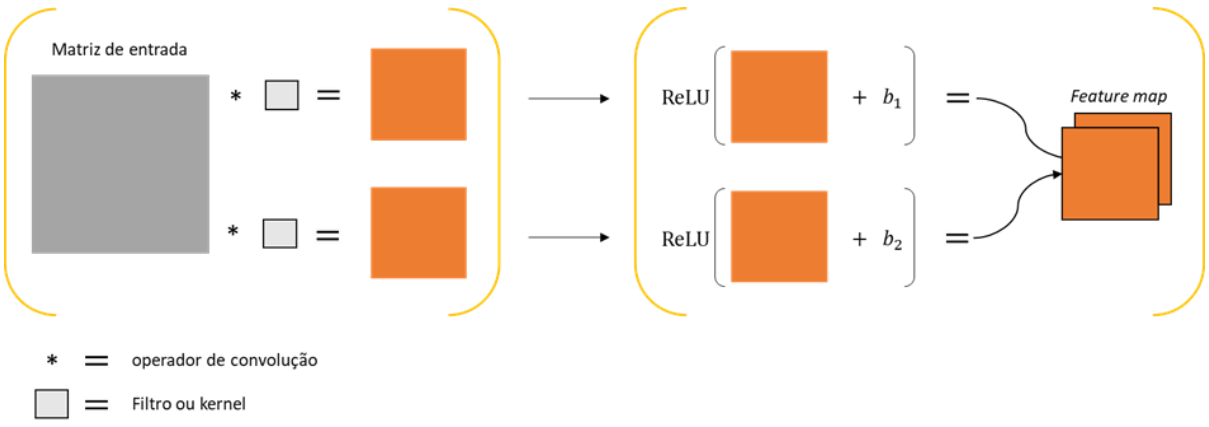


Figura 6 - Processo genérico de extração de um *feature map*

Referente ao comportamento do procedimento de convolução, tem-se, além das dimensões do kernel, dois parâmetros importantes para seu comportamento e, consequentemente, sua matriz resultante, chamados *stride* e *padding*. Na janela deslizante, o filtro, após escanear determinada região de pixels, migra para a próxima região. Nesta migração ele pula determinada quantia de pixels, procedimento intitulado *stride*. Para maior controle sobre as dimensões da matriz resultante pode-se utilizar, além do *stride*, o *padding*, o qual adiciona zeros ao redor da matriz. Este último parâmetro é muito útil para se reduzir a taxa de subamostragem da matriz, além de forçar maior utilização de pixels nas bordas.

Adicionalmente à operação de convolução, tem-se a operação de *pooling*. Comportando-se igual a uma janela deslizante, o filtro *pooling* extrai a média ou a máxima das regiões pela qual passa, entretanto sem a fase de parametrização, mantendo a quantia de canais original, no intuito de fortalecer a propriedade da invariância de tradução.

Algumas das CNNs mais populares abrangem AlexNet, VGGNet e, explorada mais adiante, ResNets. Acoplada à rede neural tem-se, geralmente, RNAs, a quais usufruem das *features* finais para tomadas de decisões complexas como classificação, estas sendo intituladas "completamente conectadas" (FC). A Figura 7 exemplifica uma rede CNN genérica e suas correspondentes camadas FC para, por exemplo, classificação entre 10 classes.

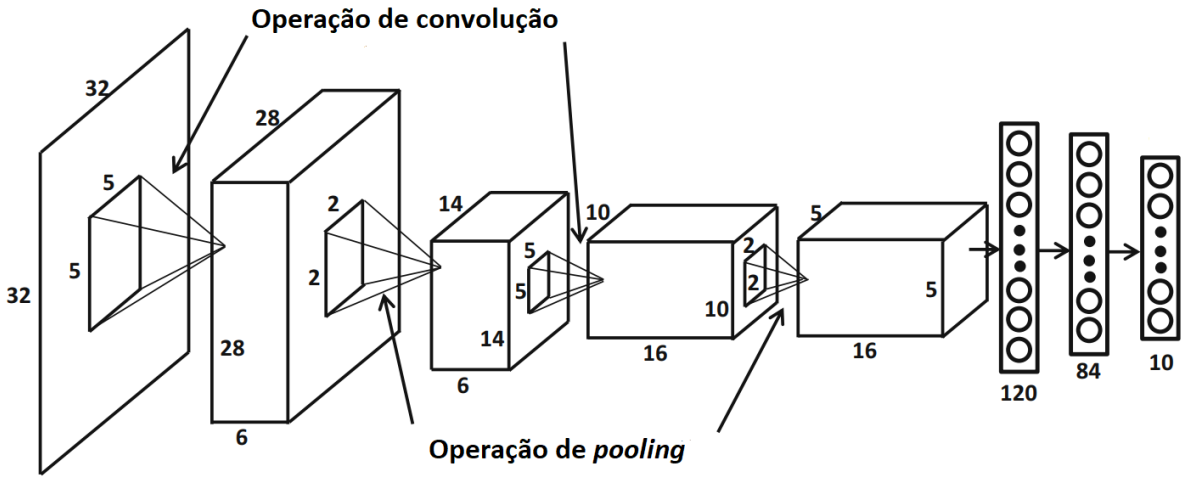


Figura 7 - Rede Neural Convolucional genérica

Fonte: Aggarwal (2018) (modificado)

3.2.3 Aprendizado Profundo

Conforme exposto nos itens acima, as RNAs e CNNs operam sobre níveis, também intituladas camadas e, dependendo da profundidade de tais níveis, as tarefas as quais pode-se endereçar são limitadas, possivelmente sendo mais eficientemente abordadas por outros algoritmos. Diante da profundidade da rede pode-se classificá-la entre raso ou profundo, e ainda conforme tal profundidade pode-se endereçar problemas de menor a maior complexidade. Daí tem-se o conceito de Aprendizado Profundo.

3.2.4 Aprendizado Profundo e Sensoriamento Remoto

As técnicas do Aprendizado Profundo vêm ganhando espaço no campo do Sensoriamento Remoto, conforme se expande a aquisição e disponibilidade de dados. Entretanto, Zhu *et al*. (2017) aponta para alguns desafios inerentes ao campo, algumas aqui abordadas:

- Desde dados matriciais até vetoriais, em sensoriamento remoto compreende ampla gama de tipologias de dados. Na aquisição de imagens ópticas o usuário tem a seu dispor ampla variedade de resoluções espaciais, espectrais e radiométricas, além de sensores como Radares de Abertura Sintética (SAR), os quais resultam em geometrias e conteúdo completamente diferentes. Neste quesito faz-se necessário metodologias capazes de usufruir destas diferentes formas de representação do mundo real, independentemente ou sinergicamente.
- Dados de sensoriamento remoto são georreferenciados. Para dados matriciais, cada elemento possui sua correspondente coordenada geográfica, permitindo a combinação destes com outras formas de informação também georreferenciadas como as camadas em um SIG. Neste sentido, tem-se a oportunidade de utilizar tais relações de forma aditiva, ou alternativa, na solução de problemas.
- A variável temporal é especialmente importante para dados de sensoriamento remoto. Faz-se público repositórios de dados capturados em cada vez menores intervalos de tempo. Desta forma a análise temporal toma diferente importância no entendimento de processos físicos e sociais, abrindo a oportunidade de adaptação e utilização de algoritmos de Aprendizado Profundo especializados na extração de informações temporais, como Redes Neurais Recorrentes (Aggarwal, 2018).
- Atualmente, a escala na qual se dá a captura de dados do sensoriamento remoto se traduz em centenas a milhares de terabytes gerados por ano, portanto a necessidade de

algoritmos eficientes e facilmente adaptáveis para aplicação em diferentes regiões da superfície terrestre se faz cada vez mais presente.

- Tradicionalmente, o campo do sensoriamento remoto busca o entendimento de processos geológicos, biológicos, físicos e químicos, como composição mineral do solo, concentração de gases na atmosfera e movimentação de terra, além das tarefas discutidas neste trabalho como classificação e segmentação na produção de mapas de uso e cobertura do solo. Tais processos denotam conhecimento prévio do especialista a respeito do problema, antes de qualquer processo de estimação. Assim se coloca em xeque a premissa principal do Aprendizado Profundo, o qual emprega um sistema em que se desconhece as variáveis intrínsecas.

3.3 VISÃO COMPUTACIONAL (CV)

O campo da Visão Computacional (CV) aborda um amplo espectro de problemas, geralmente interdisciplinares, entretanto todos em volta do entendimento do mundo real através de imagens digitais (Szeliski, 2021). Os problemas como Reconhecimento Óptico de Caracteres (OCR), modelagem 3D e reconhecimento facial são assuntos clássicos na área. As formas em que se atinge tais proposições são diversas, com a utilização de conceitos e métodos de ML e DL atualmente liderando fortes progressos através da utilização de RNAs, CNNs e outros algoritmos. No que concerne a análise de imagens orbitais e aéreas, a classificação, segmentação semântica e, especialmente, detecção de objetos são tarefas naturais, as quais podem ser abordadas com excelência através de algoritmos comuns à área da Visão Computacional.

3.3.1 Detecção de Objetos (OD)

Diversos autores argumentam a tarefa de Detecção de Objetos (OD) ser essencialmente mais complexa do que classificação e segmentação semântica (Carvalho *et al.*, 2021; Szeliski, 2021; Zhao *et al.*, 2018).

O OD remete em não só localizar independentemente cada alvo em uma imagem, como classificá-lo dentre potenciais classes. O problema da localização em si já desfruta de amplos debates e apenas recentemente vem sendo tratada com sucesso.

Dentre as metodologias empregadas para a tarefa de OD no meio de Sensoriamento Remoto pode-se divisar 4 categorias predominantes: métodos baseados em *Template Maching*, baseados em conhecimento, através de OBIA (ou GEOBIA) e, por fim, métodos baseados em

Aprendizado de Máquina, estes não sendo necessariamente mutuamente excludentes (Cheng e Han, 2016) Como já mencionado, métodos baseados no Aprendizado de Máquina vêm demostrando desempenho superior aos demais, especificamente modelos do Aprendizado Profundo, abordando o problema como um problema de classificação e usufruindo de sua robusta representação de *features* e classificadores. Alguns dos algoritmos clássicos de ML para representação de *features* são: Histogramas de Gradientes Orientados (HOG) e bolsa-de-palavras (BoW), enquanto para classificadores são Máquinas de Suporte a Vetor (SVM) e RNAs.

Dentre as metodologias de ML consideradas estado-da-arte para a tarefa de OD, pode-se divisar duas principais vertestes: métodos baseados em regressão e métodos baseados na proposição de regiões, ou de duas etapas. Métodos baseados em regressão são relativamente mais simples e eficientes por se tratar de apenas um estágio, alguns dos mais famosos incluindo OverFeat (Sermanet *et al.*, 2013), You Only Look Once (YOLO) (Redmon *et al.*, 2015) e Single Shot Detector (SSD) (Liu *et al.*, 2016). Utilizando o paradigma de reconhecimento por regiões, o método R-CNN (Girshick *et al.*, 2013b) iniciou uma família de modelos os quais dividem o processo de detecção em duas etapas: primeiro se propõe regiões de interesse (RoI) para depois classificá-los e refiná-los.

Entretanto, a implementação de tais metodologias em imagens de Sensoriamento Remoto vêm sendo difundida apenas recentemente com a disseminação de grandes quantias de imagens de alta resolução espacial, acompanhadas de anotações, como vias e edifício, fruto da recente evolução de sensores de imageamento, armazenamento de metadados e em nuvem etc. Não obstante, por mais recente que seja, a exploração de técnicas de ML para aplicação em imagens orbitais e aéreas já apresenta promissores resultados em tarefas de grande complexidade como Super-Resolução (Courtrai, Pham e Lefèvre, 2020) e OD (Zhao *et al.*, 2017). Alguns dos repositórios públicos mais populares envolvem: SpaceNet, UC-Merceds e DOTA.

Outros tópicos são igualmente importantes e vem demonstrando grandes avanços se tratando de metodologias ML no Sensoriamento Remoto, como análise de imagens hiperespectrais e de Radar de Abertura Sintética (SAR) (Cheng e Han, 2016).

3.4 ARQUITETURA MASK R-CNN

Para entendimento da arquitetura Mask R-CNN precisa-se primeiro do entendimento básico da família de arquiteturas R-CNN. Proposta por Girshick, Ross *et al.* (2014), a arquitetura R-CNN aborda a tarefa de OD utilizando duas premissas principais: aplicação de Redes Neurais

Convolucionais de alta capacidade para a extração de RoIs e, para cenários de limitada disponibilidade de dados, utilização de pesos pré-treinados seguido de, possivelmente curto, treinamento para o objetivo final específico, metodologia também conhecida como *transfer learning*.

3.4.1 R-CNN

Uma abordagem relativamente simples em OD utilizando CNNs é o algoritmo de "janelas móveis", aonde a CNN escaneia a imagem em pequenos trechos por vez, inferindo a localização e classificação por trecho. Entretanto, tal abordagem apresenta diversos problemas como maior exigência de computação, cada trecho potencialmente encobrindo diversos objetos e/ou enquadrando apenas parcialmente o objeto.

Buscando maior precisão em comparação às abordagens de regressão do período e ainda evitando os entraves mencionados, os autores propuseram um sistema dividido em três módulos: o primeiro encarregado da proposição de RoIs, os quais são então "alimentados" ao segundo módulo, uma CNN que extrai o *feature map* correspondente de cada RoI, cada qual passando pelo terceiro e último módulo, um conjunto de SVMs encarregados da classificação.

O módulo de proposição de RoIs é encarregado por um algoritmo intitulado busca seletiva, entretanto, segundos os autores, o algoritmo específico utilizado é agnóstico ao modelo, portanto poderia ter-se utilizado outros algoritmos. A implementação também introduz um aparato de refinamento do RoI através de regressão. A Figura 8 exemplifica a arquitetura. Os autores relataram desempenho significativamente maior da arquitetura em relação aos modelos previamente divulgados, todos treinados no mesmo repositório de dados.

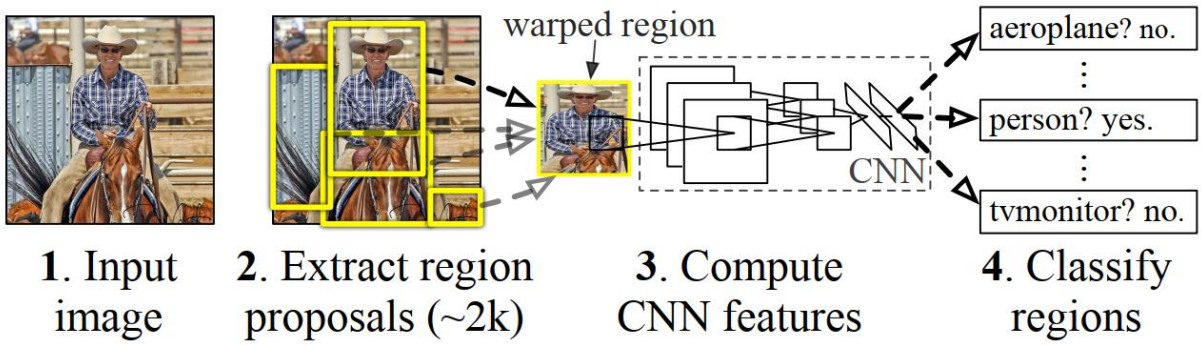


Figura 8 - Arquitetura R-CNN

Fonte: Girshick (2013)

O modelo R-CNN, entretanto, demandava muito tempo de processamento, pois cada região extraída passava pela CNN e seu resultado logo em seguida pela SVM, além de comportar complicado esquema de treinamento. A arquitetura Fast R-CNN, também proposta por Girshick (2015), teve o intuito de solucionar o problema introduzindo o algoritmo RoI *pooling*, compartilhando grande parte da computação necessária para as proposições, além de simplificar a etapa de treinamento.

A arquitetura Fast R-CNN, apresentada na Figura 9, estende a ideia de uma outra arquitetura, chamada SPPnet (He *et al.*, 2014), também proposta para solucionar a custosa etapa de computação. A ideia era, após etapa de proposição de RoIs, alimentar a imagem como um todo à CNN encarregada da extração das *features* e, então, extrair diretamente do *feature map* obtido as regiões referentes aos RoIs propostos.

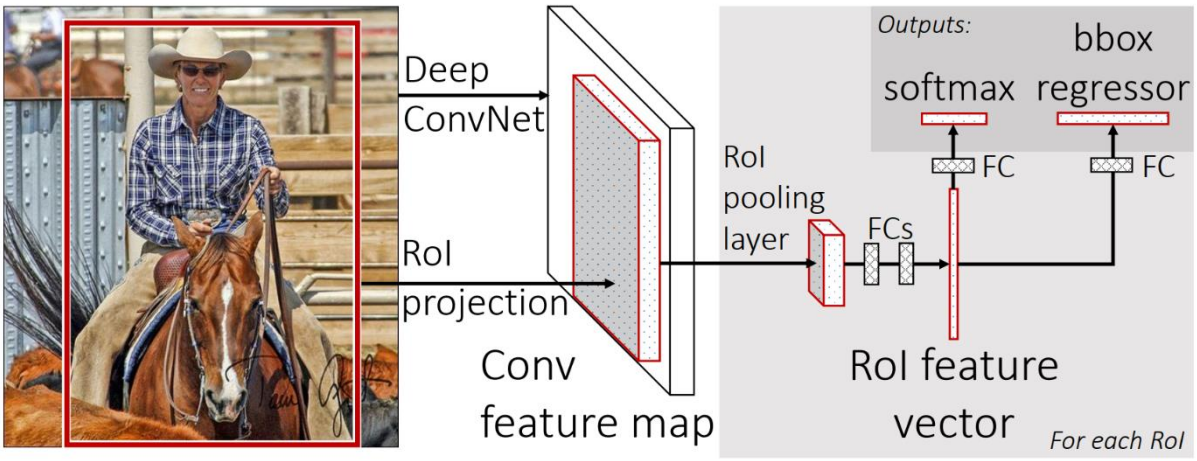


Figura 9 - Arquitetura Fast R-CNN

Fonte: Girshick (2018)

Desta forma não são necessárias várias passagens através da CNN, compartilhando a computação para todas as regiões. Diferentemente da arquitetura SPPnet, a Fast R-CNN simplifica a estrutura introduzindo uma camada intitulada RoI Pooling, encarregada da extração da RoI diretamente do *feature map*, além de utilizar camadas FC a qual se divide em duas camadas irmãs: a primeira encarregada da classificação e a segunda do refinamento das dimensões do RoI. Há de notar que a etapa de proposição de RoIs ainda é de acordo com um algoritmo "a parte", a qual também demanda, relativamente, considerável tempo de processamento.

A desvantagem de se utilizar algoritmos desconectados do processo de treinamento do modelo é que este não possui potencial de refinamento de acordo com cada tarefa específica, portanto,

apesar de comportamento previsível, é também limitado. Segundo Ren (2016), a arquitetura Faster R-CNN, a qual completa a família clássica de R-CNNs propõe um modelo completamente conectado. A nova arquitetura traz consigo a *Region Proposal Network* (RPN), encarregada da proposição de RoIs, usufruindo da premissa que a proposição de RoI pode também utilizar do *feature map* extraído para inferência.

Ao contrário de esforços anteriores os quais, para se encarregar de objetos de diferentes escalas e proporções, repetiam a imagem ou os filtros com distintos aspectos (Ren *et al*., 2016), os autores propõem uma estrutura chamada âncora. As âncoras servem como RoIs hipotéticos predefinidos e o objetivo final da RPN é selecionar as melhores candidatas a possuir um objeto, além de refinar suas dimensões. Isto se dá da seguinte maneira: o *feature map*, obtido da imagem, possui um ponto de âncora centrado em cada pixel; a RPN opera sobre cada ponto de âncora em um comportamento semelhante a janelas deslizantes: fragmento por fragmento, tomando o pixel central e alguns ao seu redor, a RPN, dotada de uma camada convolucional e duas FCs irmãs, mapeia o fragmento para um vetor de menor espaço vetorial, o qual é entregue para o par de camadas FC, cada qual responsável por: determinar, para cada âncora, a probabilidade desta possuir ou não um objeto de interesse e, se houver alta probabilidade de possuir, qual o melhor ajuste da âncora para melhor enquadrar o objeto. As camadas FC finais ditam efetivamente a quantia de âncoras a se dispor. Para o caso em que se dispõe de *k* âncoras, de diferentes escalas e proporções, tem-se 2x*k* probabilidades a se determinar na camada de classificação e 4x*k* ajustes na camada de ajuste do enquadramento da âncora, referentes à coordenadas x e y além de altura e comprimento da âncora, coordenadas parametrizadas nos formatos expostos pelas Equações (3.6), (3.7), (3.8) e (3.9). A Figura 10 exemplifica a estrutura discutida.

$$t_x = \frac{(x - x_a)}{w_a} \tag{3.6}$$

Onde:

t_x = parametrização do ajuste da coordenada x

x = coordenada x inferida

x_a = coordenada x da âncora

w_a = comprimento da âncora

$$t_y = \frac{(y - y_a)}{h_a} \tag{3.7}$$

Onde:

t_y = parametrização do ajuste da coordenada y

y = coordenada y inferida

y_a = coordenada y da âncora

h_a = altura da âncora

$$t_h = \log\left(\frac{h}{h_a}\right) \tag{3.8}$$

Onde:

t_h = parametrização do ajuste da altura

h = altura inferida

h_a = altura da âncora

$$t_w = \log\left(\frac{w}{w_a}\right) \tag{3.9}$$

Onde:

t_w = parametrização do comprimento

w = comprimento inferida

w_a = comprimento da âncora

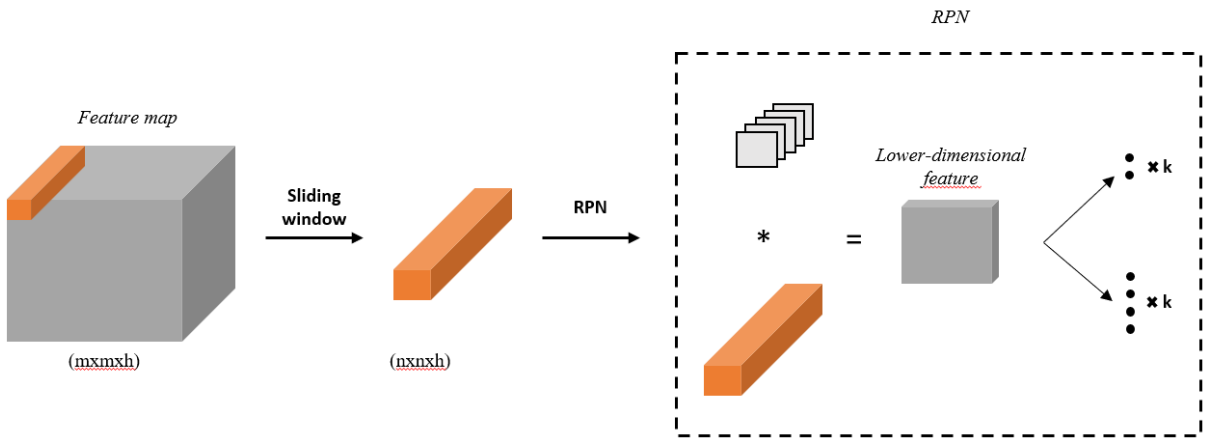


Figura 10 - *Region Proposal Network*

A parametrização exposta acima representa o esforço da regressão não de inferir as coordenadas em si, e sim um deslocamento das coordenadas da âncora na tentativa de melhor se aproximar do real. Nesta premissa a função perda toma forma exposta pela Equação (3.10):

$$L(\{p_i\},\{t_i\}) = \left(\frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*)\right) + \left(\lambda \frac{1}{N_{reg}}\sum_i p_i^* \times L_{reg}(t_i, t_i^*)\right) \qquad (3.10)$$

Onde:

i = índice da âncora

p_i = probabilidade da âncora i possuir um objeto

p_i^* = *ground-truth* se a âncora enquadra ou não um objeto

t_i = vetor representando o ajuste do enquadramento

t_i^* = vetor representando o enquadramento *ground-truth*

L_{cls} = função erro para a classificação

L_{reg}= função erro para a regressão do ajuste das âncoras

N = termo regularizador

λ = peso da regularização

As equações acima possuem importante papel no entendimento da arquitetura Mask R-CNN, pois esta pode ser entendida como uma extensão da arquitetura Faster R-CNN. De fato, a arquitetura Mask R-CNN concebida por He, Kaiming *et al.* (2018) propõe a adição de um trecho na estrutura, paralelo aos mecanismos de localização e classificação "originais", para inferência

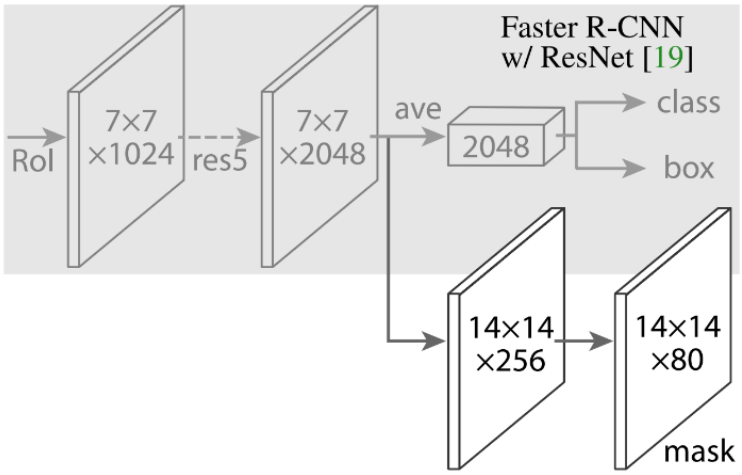


Figura 11 - *Mask branch*

Fonte: He (2017)

de uma máscara de segmentação, mais especificamente, para segmentação de instâncias, como pode-se observar na Figura 11, além da substituição da camada RoI Pooling para outra a qual o autor intitula RoI *Aling*.

3.4.2 *RoI Align*

A CNN utilizada para extração do *feature map* na etapa inicial do modelo é também intitulada de *backbone*, pois esta pode ser modifica ou até completamente substituída, ditando apenas a qualidade das features extraídas, relativo ao objetivo final, e as dimensões do feature map. De acordo com o *backbone* utilizado, a taxa de subamostragem varia, significando diferença nas dimensões, tanto de comprimento e altura como de "profundidade" do *feature map*. Invariavelmente às estas dimensões, as RoI propostas são referentes aos pixels da imagem, portanto a taxa de subamostragem deve ser aplicado conjuntamente às proposições. Neste redimensionamento ocorre o fenômeno de quantização. Supondo uma imagem *n*x*n* e uma taxa de subamostragem de 32, o *feauture map* terá dimensões $\frac{n}{32} \times \frac{n}{32} \times n° \, de \, canais$. O tamanho específico das amostras pode ser manipulado, agora supondo uma RoI de dimensão *h*x*w*, a razão $\frac{h}{32} \times \frac{w}{32}$ pode resultar em números não inteiros.

O RoI polling contorna este problema arredondando-os, portanto, as dimensões hipotéticas 2,45x2,89 seriam convertidas em 2x2, daí acarreta-se o fenômeno de quantização, significando a perda de informação. Considerando que cada pixel do *feature map*, neste exemplo, representa 32 pixels da imagem, tem-se um erro de alinhamento de, em torno, 14 e 28 pixels entre a RoI no *feature map* e na imagem original. Há também quantização após este procedimento, onde há novamente redimensionamento do RoI em uma matriz de dimensão fixada antes de passar adiante na rede.

Os autores da arquitetura Mask R-CNN demonstram que, tratando-se apenas de Detecção de Objetos, o desempenho não é largamente afetado, entretanto, para a segmentação de instâncias, o alinhamento do RoI entre *feature map* e imagem deve ser preciso para resultados satisfatórios (He *et al.*, 2017). Portanto os autores propõem a utilização da camada RoI Align, a qual funciona da seguinte maneira: a RoI inferida é subdividida em 4 células igualmente espaçadas, cada qual com 4 pontos de amostragem; os pontos de amostragem são utilizados para computar os valores exatos do *feature map* utilizando interpolação bilinear; por final agrega-se os valores tirando a média, ou a máxima, dos valores encontrados nos pontos de amostragem. Desta forma se evita a quantização tanto na etapa de espelhamento do RoI da imagem para o *feature map*

como seu posterior redimensionamento para uma matriz de dimensões fixas. A Figura 12 exemplifica o mecanismo.

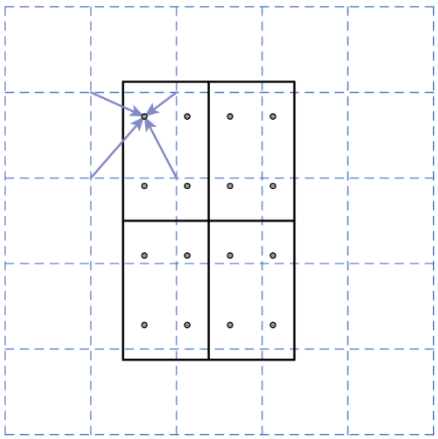


Figura 12 - Operação do RoI *Aling*

Fonte: He (2017)

3.4.3 *Mask Branch*

Dentre as metodologias utilizadas no ML para a tarefa de segmentação semântica, Redes Completamente Convolucionais (FCN) são amplamente difundidas. Muito próximas de CNNs clássicas, a FCN, ao invés de utilizar RNAs no final de suas arquiteturas para tarefas como classificação, utiliza de camadas convolucionais para executar o *upsampling*, retornando as dimensões do *feature map* aos da imagem original de forma que, ao final, tem-se uma matriz de mesma, ou aproximada, altura e comprimento porém de pixels catalogados entre n classes. O trecho adicionado à arquitetura Faster R-CNN utiliza do mesmo mecanismo, como pode-se observar na Figura 11, entretanto em RoIs pré-selecionados contendo, teoricamente, apenas um objeto.

3.5 TRABALHOS SIMILARES

Em meio às diversas publicações científicas no campo do Sensoriamento Remoto, a fração a qual foca na utilização de tecnologias do Aprendizado Profundo é gradativamente maior (Figura 13). A partir de análise metódica da literatura, os trabalhos de Cheng e Han [s.d.], Li *et al*. [s.d.] e Zhu *et al*. (2017) destacam os seguintes tópicos do Sensoriamento Remoto os quais foram sujeitos a aplicações de Aprendizado Profundo: análise de imagens hiperespectrais, interpretação de imagens de Radar de Abertura Sintética, interpretação de imagens orbitais de alta resolução, fusão de dados multimodais, reconstrução 3D, classificação de imagem, segmentação semântica e detecção de objetos.

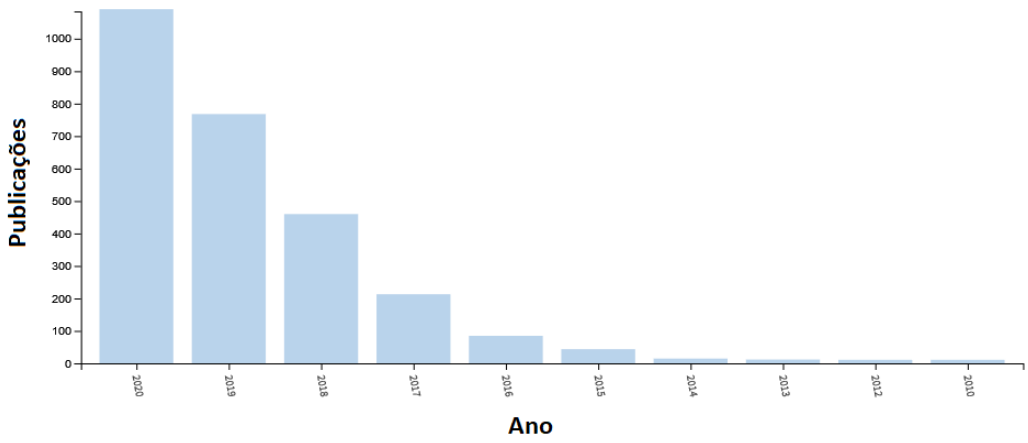


Figura 13 - Publicações referentes ao assunto "*Remote Sensing Deep Learning*" de 2010 a 2020

Fonte: Web Of Science (2021)

Referente à classificação de imagem, no campo do Aprendizado de Máquina este pode ser entendido como a rotulação de uma imagem, invariavelmente da quantia ou variedade dos objetos, diferentemente da rotulação pixel a pixel, comumente denominada segmentação semântica. Em relação a esta classificação, Pritt e Chern (2020) implementam um modelo baseado em CNN e RNA, semelhante à exemplificada na Figura 7, o qual eles afirmam demostrar superior desempenho na tarefa de classificação em imagens de satélite. Outros trabalhos como o de Castelluccio *et al.* (2015) também demostra excelência na tarefa de classificação referente ao Uso e Cobertura do Solo, alguns ampliando o objetivo para a detecção de mudanças no Uso e Cobertura do Solo (Cao, Dragicevic e Li, 2019).

As CNNs fazem-se presentes na grande maioria das aplicações referentes a imagem no Aprendizado Profundo, tanto em classificação como segmentação semântica (Zhu *et al.*, 2017). Referente a segmentação semântica, entre as arquiteturas as quais demonstram forte desempenho estão as Redes Completamente Convolucionais (FCN) (Long, Shelhamer e Darrell, 2014). Batista (2019) demonstra a capacidade da rede intitulada U-Net, considerada uma FCN, na segmentação de vias na cidade de São Luís, MA.

Os trabalhos de Cheng e Han (2016) e Li *et al.* (2019) apontam para uma tendência de rápido crecimento na utilização de modelos do Aprendizado de Máquina para a tarefa de Detecção de Objetos aplicada a imagens do Sensoriamento Remoto para os alvos mais diversos como vias, edifícios, automóveis, árvores etc. Trabalhos recentes como os de Zhao *et al.* (2017) e Mohanty *et al.* (2020) demonstram a acurácia de modelos baseados na proposição de regiões para a

detecção de edificações e posterior segmentação. Tais avanços permitem a elaboração de mapas de uso e cobertura do solo com maiores níveis de detalhamento entre as classes temáticas, futuramente possibilitando a contagem de edificações no território, área concretada e cobertura arborizada.

4. METODOLOGIA

Para a aplicação da arquitetura Mask R-CNN é necessário o desenvolvimento do código encarregado da computação, estruturação de um banco de dados, treinamento e avaliação de desempenho do modelo e, por final, desenvolvimento de uma interface amigável capaz de implementá-lo na constituição de napas temáticos de uso e cobertura do solo. O fluxograma exposto pela Figura 14 demonstra a metodologia proposta.

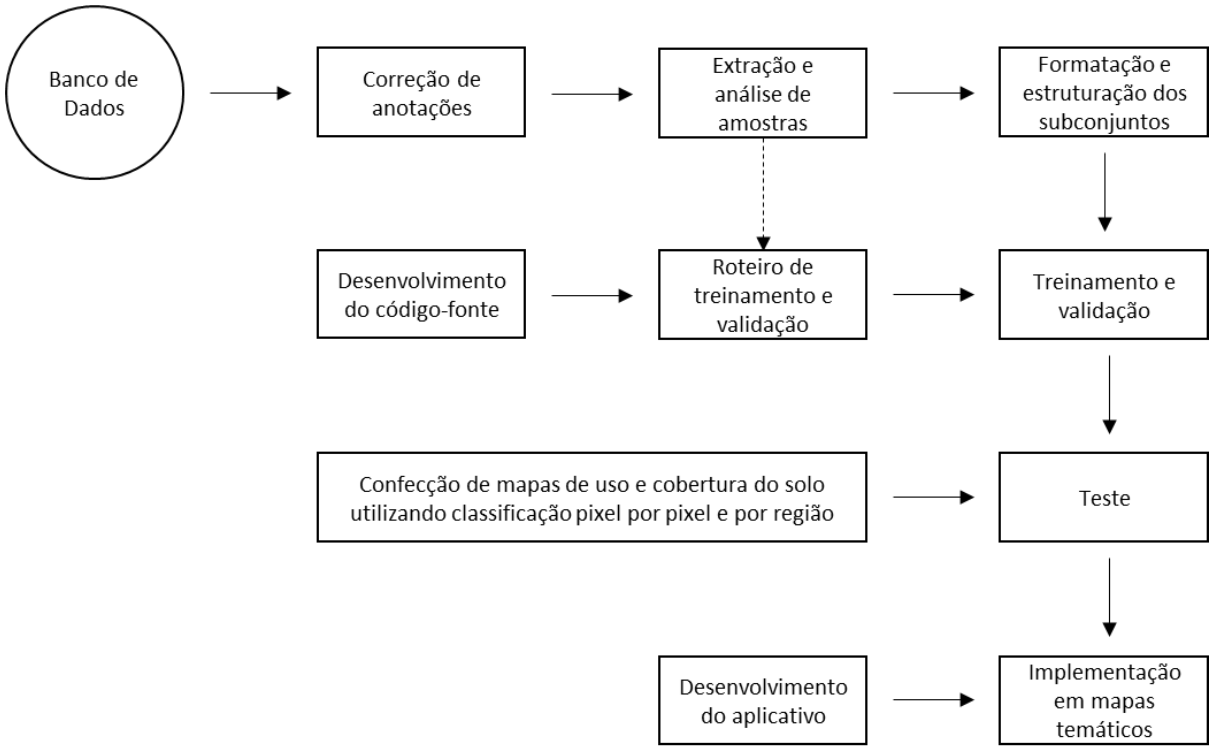


Figura 14 - Fluxograma da metodologia

4.1 MATERIAIS

O conjunto de dados disponíveis para este trabalho é composto por imagens ortorretificadas de 24 cm de resolução espacial da região demarcada na Figura 15 pertencente ao DF, capturadas em meados de 2018 e 2019 a serviço da Terracap, além de dados vetoriais de edificações da mesma região constituídos através de interpretação visual e digitalização manual no mesmo período.

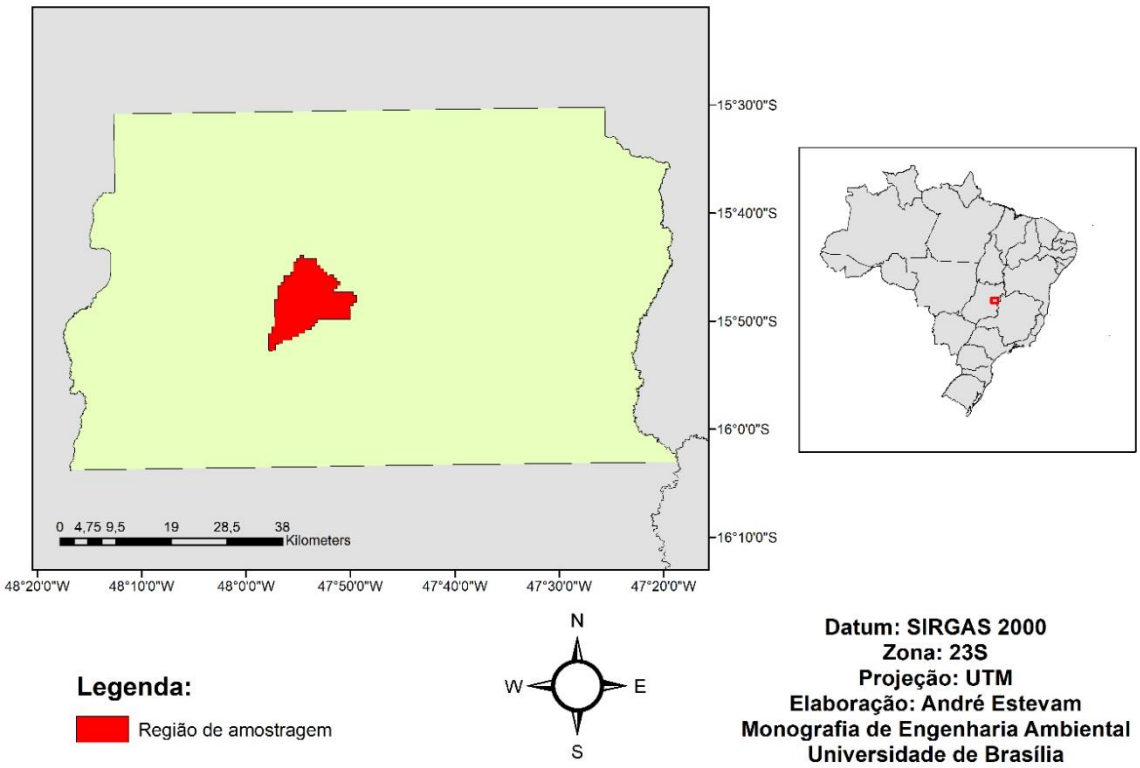


Figura 15 - Região de amostragem

Adicionalmente se utilizará de uma fração do repositório público de dados, o SpaceNet, curado por Zhao *et al.* (2018), composto por imagens de 30 cm de resolução espacial de diversas localidades como Paris, Vegas etc., conjuntamente aos correspondentes dados vetoriais a respeito de edificações e vias.

Para manipulação das imagens aéreas e dados georreferenciados se utilizará dos ambientes computacionais SIG: ArcGIS, QGIS e Spring. Para formatação e análise das amostras, além do desenvolvimento, treinamento, validação e teste do modelo, se utilizará da linguagem de programação *Python*, conjuntamente ao *framework* PyTorch, o qual é uma biblioteca de códigos especializado em algoritmos de Aprendizado de Máquina. Os *scripts* serão confeccionados através do ambiente de produção de códigos Notepad++.

Para ambiente de teste do código e treinamento do modelo, visto exigência de alto poder de processamento se utilizará o Google Colab, o qual disponibiliza uma GPU, porém por tempo limitado.

Para o desenvolvimento da interface amigável integrada a um ambiente SIG se utilizará do módulo Qt em associação ao ambiente SIG QGIS, o qual permitirá a execução de um *plugin* no qual o usuário terá acesso à inserção da imagem, processamento e exportação do resultado do modelo.

4.2 BANCO DE DADOS

4.2.1 Correção de anotações

Devido a discrepâncias nos períodos de constituição dos dados vetoriais e captura das imagens do DF é necessário garantir a mínima quantia possível de edificações sem anotação, além de anotações incorretas, evitando impacto no processo de treinamento. Para tanto uma inspeção completa é necessária.

4.2.2 Estruturação das amostras

Para treinamento do modelo é necessário a extração de um conjunto de amostras. Cada amostra é composta por um par de matrizes n-dimensional: uma imagem e correspondente dados vetoriais. Supondo uma imagem de 300x300 pixels, o formato dos dados vetoriais deve ser reestruturado em uma matriz de 300x300xnúmero de instâncias (edificações), onde cada região referente a um edifício deve possuir um valor único e 0 nas demais células, devido a necessidade de identificar cada alvo independentemente.

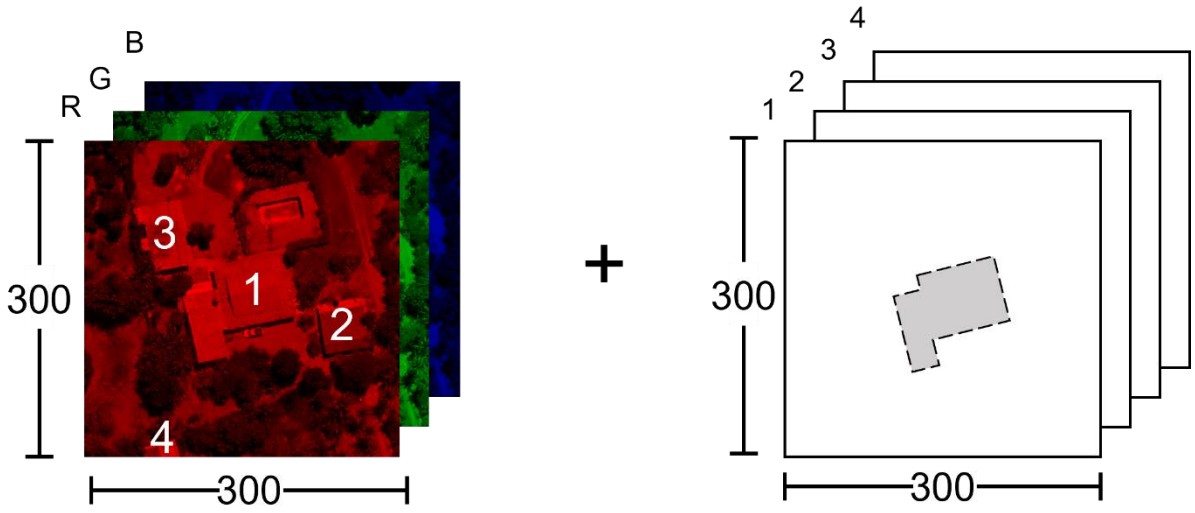


Figura 16 - Amostra ilustrativa

Tal procedimento será realizado através da ferramenta ArcGIS em conjunto à linguagem *python*. Um exemplo ilustrativo de amostra é exposto na Figura 16.

Após extração das amostras, estas serão analisadas para aferir a distribuição do tamanho através de todas as instâncias, além da distribuição da quantia destas por amostra, utilizando tais informações para auxiliar na definição de alguns componentes do modelo e roteiro de treinamento (hiperparâmetros).

O conjunto de amostras será dividido nos seguintes subconjuntos: treinamento, o qual será utilizado para ajustar os parâmetros do modelo, validação, utilizado para determinação de valores ótimos dos hiperparâmetros, e teste, o qual será utilizado para a avaliação dos valores de métricas de precisão do modelo na tarefa proposta.

4.3 MODELO

Após a estruturação das amostras parte-se para o ajuste dos parâmetros do modelo, seu treinamento. Aqui se dá um processo iterativo: treinamento, validação e ajuste "manual" dos hiperparâmetros até a etapa final de teste de desempenho. Para porção considerável dos hiperparâmetros repete-se o utilizado no trabalho o qual propôs o modelo; entretanto, outros, como escala das âncoras e taxa de treinamento (Lr), dependem da tarefa a ser abordada, quantia de amostras, dimensões e características da amostra, algoritmos utilizados etc.

Usualmente utiliza-se como *backbone* a rede ResNet50 ou a ResNet101, mencionadas no item 3.2.2, dependendo do *hardware* disponível à pesquisa e objetivo do modelo; entretanto, este Projeto Final utilizará da ResNet50 invariavelmente das recomendações. Seria de interesse testar o desempenho do modelo empregando a rede ResNet101 por esta conter rede mais extensa de camada e, portanto, apresentar melhor capacidade na extração de *features* complexas; entretanto, tal emprego tomaria muito mais tempo de treinamento e o ambiente de execução de código Google Colab não permite execuções maiores que, aproximadamente, 36 horas, limitando significativamente a quantia de épocas que comporiam o treinamento.

4.3.1 Roteiro de Treinamento e Validação

Como discutido no item 3.2, modelos supervisionados como o Mask R-CNN necessitam de grandes quantias de dados catalogados para desempenhos satisfatórios. Para diversas aplicações, entretanto, não se dispõe de volume suficientemente grande de dados para o objetivo final. Para se contornar tal entrave pode-se usufruir do fato de que, em camadas mais iniciais da rede, a informação computada é de cunho mais simples, como contornos, orientações etc., possivelmente sendo útil para diferentes objetivos. Diante desta perspectiva, modelos de

Aprendizado Profundo comumente iniciam o treinamento a partir de pesos pré-treinados, visando o compartilhamento de parâmetros entre modelos espelhados na mesma arquitetura, porém de diferentes objetivos finais, na esperança de que as *features* capturadas por um sejam minimamente similares às *features* "ótimas" para o objetivo do outro. Além da utilização de pesos pré-treinados este trabalho fará proveito é o *transfer learning*. Utilizando da mesma premissa, o *transfer learning* visa o treinamento da rede utilizando outro repositório, potencialmente maior, de amostras similares às amostras as quais se quer extrair a estatística, visando o aproveitamento de *features* extraídas em camadas iniciais. Terá, portanto, efetivamente 2 treinamentos: o primeiro utilizando o repositório público mencionado no item 4.1 e o segundo utilizando as amostras extraídas do Distrito Federal.

À disposição do treinamento tem-se alguns métodos comumente utilizados para minimização do erro e localização do mínimo global, os mais populares abrangendo Gradiente Estocástico Descendente (SGD), Adam e RMSProp. Este trabalho não visa uma análise pormenorizada de cada método, entretanto é importante a experimentação para averiguar qual melhor permite um melhor ajuste dos parâmetros aos dados disponíveis.

As métricas de precisão serão aquelas utilizadas pelo *Common Objects in Context* (COCO) para OD e segmentação de instâncias, especificamente: precisão e revocação, utilizando Intersecção sobre União (IoU), uma vez que a complexidade da saída do modelo exige um método de avaliação adaptado.

4.3.2 Aplicativo

Por final, após desenvolvimento do modelo e seu treinamento, será desenvolvido uma interface amigável a qual permitirá a aplicação do mesmo de forma intuitiva. O seguinte roteiro permeia o design da interface:

1. Entrada da imagem a ser classificada e segmentada.
2. Definição do *threshold*, ou precisão, através do qual as detecções serão filtradas.
3. Opção de extração de dados vetoriais georreferenciados em formato shapefile. A saída padrão é um raster.
4. Definição do diretório e nome dos arquivos de saída.
5. Execução.

A interação interface-ambiente SIG permite fácil integração ao objetivo final deste trabalho, o qual é aperfeiçoar a confecção de mapas temáticos de uso e cobertura do solo.

5. RESULTADOS

O trabalho considera como produto final o *plugin*, o qual fará a integração do modelo com ambientes SIG, bem como a base de dados estruturada especificamente para treinamento de modelos supervisionados de aprendizado de máquina no âmbito do sensoriamento remoto e os pesos treinados para modelos de arquitetura Mask R-CNN, para detecção de edificações em classes temáticas utilizadas na produção de mapas de uso e cobertura do solo.

5.1 Dados

Os conjuntos de dados totalizaram 11.058 amostras após extração e tratamento. A divisão entre os subconjuntos deu-se da seguinte forma: 9.952 para treinamento (90%), 547 para validação (5%) e 547 para teste (5%), enquanto 12 amostras foram descartadas por não conterem instâncias. Os gráficos apresentados pela Figura 18 são resultado do processo de análise para averiguar características importantes dos dados disponíveis na decisão inicial de um ponto de partida para as escalas e tamanhos das âncoras. Na Figura 17 tem-se a exemplificação de uma amostra.

Figura 17 - Exemplo de amostra após extração e tratamento

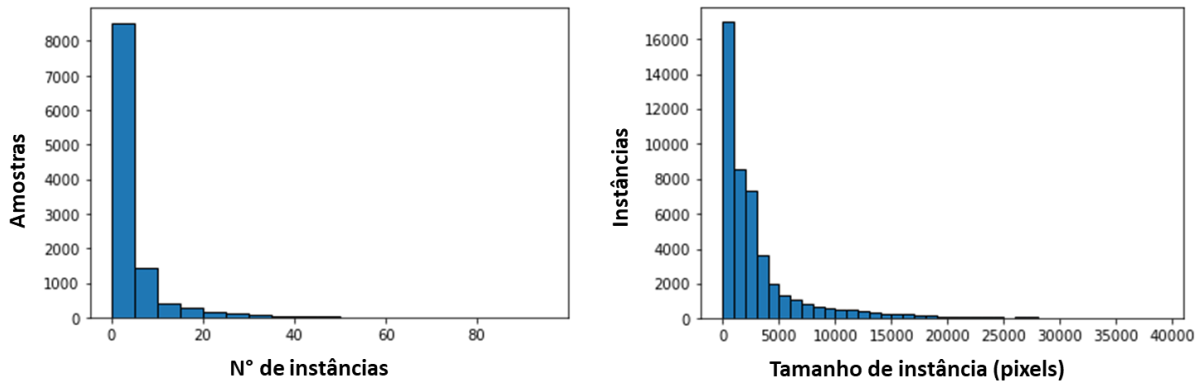


Figura 18 - Caracterização do conjunto de amostras extraídas da região do DF

Uma vez extraídos e tratados, estes devem ser corretamente lidos como entrada para seus respectivos roteiros. Tal proposição também se aplica aos dados publicamente disponíveis discutidos no item 4.1. O *script* responsável por tal tarefa está localizado no apêndice item 7.2.1.

5.2 Modelo

O modelo apresentou convergência por todo o processo de treinamento, tanto para o conjunto de amostras público como para o extraído da região do DF. A taxa de treinamento (Lr) adotado foi de 0,0001 devido à instabilidade do erro para valores maiores. Deve-se apontar que a taxa de decrescimento do erro, tanto para o conjunto de treinamento como para o de validação, permaneceu estável, o que aponta mais uma vez para o potencial contido no aumento de épocas na etapa de treinamento para um melhor ajuste dos parâmetros. Na Figura 19 pode-se observar um exemplo de entrada e saída do modelo, detalhando uma imagem que fez parte do subconjunto de amostras de teste, o qual não participou do processo de treinamento nem de validação. As métricas de precisão do modelo se encontram no apêndice de item 7.3. A instanciação do modelo foi executada pelo fragmento de código localizado no apêndice item 7.2.2.

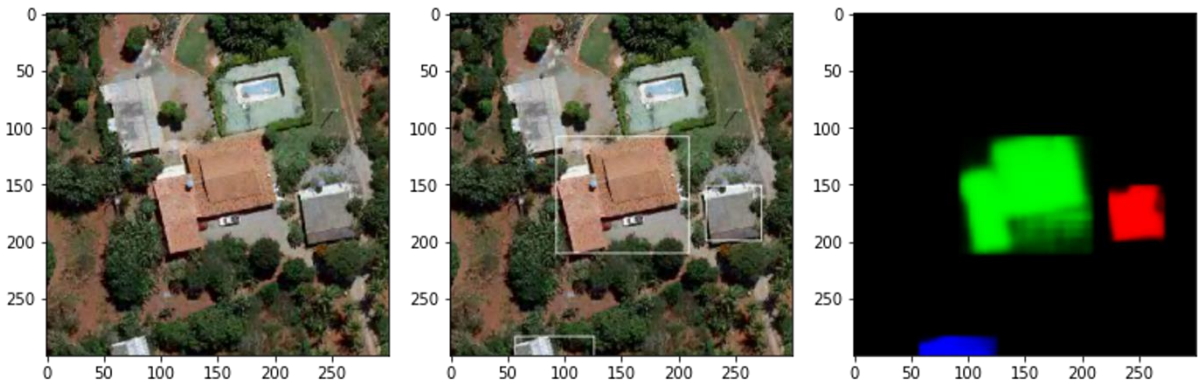


Figura 19 - Exemplo de entrada e saída do modelo

Os hiperparâmetros os quais o modelo apresentou melhor desempenho segundo o subconjunto de validação estão localizados na Tabela 1. Tais hiperparâmetros, portanto, representam a estrutura geral do modelo a ser empregado pelo *plugin* e do roteiro de treinamento. O emprego da rede neural convolucional ResNet50 como *backbone* foi feito em razão das limitações de *hardware*, como mencionado no item 4.3.

Tabela 1 - Hiperparâmetros ótimos

Hiperparâmetro	Valor
Backbone	ResNet50+fpn
Tamanho das âncoras (pixel)	[8, 16, 32, 64, 128]
Escala das âncoras	[1, 1.5, 2]
Épocas	5
Taxa de treinamento	0,00001
Camadas treinadas	Todas

Em prol de uma comparação simples e eficiente em relação ao desempenho do modelo foi executado o método clássico de classificação supervisionada por pixel Máxima Verossimilhança (MAXVER), assim como segmentação seguida por classificação por região utilizando outro método clássico, o Bhattacharya, no enxerto de imagem exposto na Figura 20. As classes temáticas utilizadas foram: vegetação, solo exposto, vias e edificação. Foi necessária a diferenciação entre via e edificação para buscar a equiparação dos alvos do modelo e dos demais métodos. Os resultados são expostos pela Figura 21 para MAXVER e Figura 22 para classificação por região utilizando o método Bhattacharya, exibindo apenas a classe de edificação no intuito de se ter uma comparação mais limpa enquanto os mapas temáticos completos estão expostos no apêndice item 7.1. A Figura 23 expõe o resultado para este modelo com detecções de probabilidade igual ou maior que 0,80.

Figura 20 - Enxerto de imagem para avaliação do desempenho do modelo

Figura 22 - Classificação supervisionada por pixel com o método MAXVER

Figura 21 - Classificação supervisionada por região com o método Bhattacharya

Figura 23 - Classificação pelo modelo de arquitetura Mask R-CNN

Os resultados acima suscitam as seguintes observações:

- O método MAXVER apresentou grande dificuldade em classificar edificação separadamente de vias, justapondo ambas na maioria das ocasiões. O modelo deste trabalho e a classificação por região apresentaram melhores resultados, apesar de ainda apresentarem falhas de detecção em relação a delimitação das áreas de edificação.
- O propósito de delimitar cada ocupação como única se apresenta mais desafiadora aos métodos clássicos de classificação, frequentemente resultando em geometrias que se estendem além dos limites da propriedade englobando duas ou mais ocupações. A classificação por região se mostra como uma solução mais interessante ao MAXVER, capturando em diversas instâncias limites verossímeis, apesar de tal solução não fornecer segmentação de instâncias, exigindo um pós-processamento. O modelo deste trabalho apresentou resultados mais refinados daquele de classificação por pixel, entretanto não tanto ao do resultado da classificação por região, apesar de oferecer segmentação por instâncias.

5.3 Aplicação

A produção do *plugin* resultou em uma série de arquivos os quais se interrelacionam para a troca de informações, processamento de dados e integração com o QGIS. Alguns dos principais componentes envolvem: a interface, os *scripts* de processamento, peso do modelo, ícone e compiladores.

A interface desenvolvida é apresentada na Figura 24, a qual segue o roteiro predefinido no item 4.3.2, compondo os seguintes elementos:

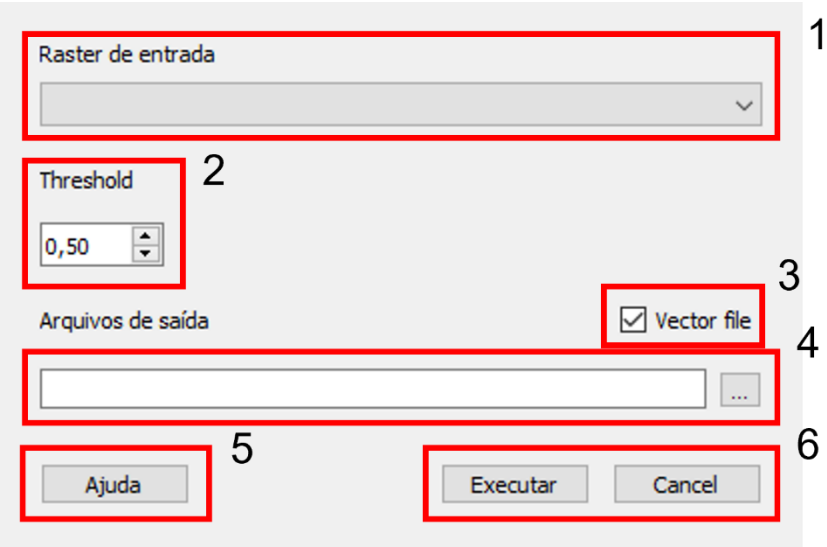


Figura 24 - Interface do *plugin*

1. Entrada de dados, ou seja, o raster o qual será repassado ao modelo. O usuário possui a alternativa de selecionar um raster já carregado na interface do QGIS ou fornecer o caminho do arquivo.
2. *Threshold*, o qual dita qual a probabilidade mínima que cada detecção deve conter, efetivamente excluindo detecções de probabilidade menor que o *threshold*.
3. Arquivo vetorial, caso o usuário queira a saída do modelo em formato vetorial através do tipo de arquivo shapefile.
4. Saída dos dados, o qual determina onde os arquivos de saída serão salvos e seus respectivos nomes.
5. Ajuda, o qual apresentará ao usuário uma janela contendo uma breve explanação do *plugin* e como utilizá-lo.
6. Executar ou Cancelar, para efetivamente executar o modelo a partir dos parâmetros fornecidos ou fechar a janela.

A interface foi desenhada visando a fácil experimentação com diferentes valores de *threshold*, uma vez que a escolha deste parâmetro possui forte impacto sobre a saída do modelo. Abaixo são expostos três resultados derivados de três *thresholds* diferentes: 0,50, 0,80 e 0,95.

Figura 25 - Saídas do modelo para diferentes *thresholds*

Por se tratar de uma segmentação de instâncias, cada conjunto de pixels representando uma detecção possuirá ID específico e, adicionalmente, acompanhará os dados de área e perímetro, caso o usuário exporte os dados em formato vetorial.

Apesar da complexidade do modelo e os processamentos que se dão em paralelo, o tempo de execução mostrou-se não só viável como melhores que os tomados pelo método MAXVER e Bhattacharya mencionados acima, uma vez que o modelo não necessita da entrada, muitas vezes manual, de amostras por parte do usuário. A Tabela 2 expõe os tempos cronometrados para três imagens de dimensões diferentes, com e sem saída de dados vetoriais, executados por uma máquina com 8 gigabytes de memória RAM e processador Inter i7.

Tabela 2 - Tempos de processamento do *plugin*

Dimensões (pixel x pixel)	Com dados vetoriais (s)	Sem dados vetoriais (s)
300x300	5,5	5,5
600x1500	6	6
3557x3751	20	15

6. CONCLUSÃO

É de crucial importância a exploração e desenvolvimento de novas técnicas e metodologias para a confecção de mapas de uso e cobertura do solo, visto a crescente complexidade do espaço urbano e rural e o dever humano de preservação da fauna e flora. Este Projeto Final de Graduação em Engenharia Ambiental representa importante passo na exploração de tecnologias emergentes aplicadas na análise de imagens do SR e a interdisciplinaridade entre os campos do Aprendizado de Máquina, Visão Computacional e Sensoriamento Remoto, visando não só o aprimoramento de técnicas, como uma tentativa de viabilização do uso da ferramenta invariavelmente ao nível técnico do usuário final.

O modelo desenvolvido apresenta resultados promissores, dando à comunidade científica, bem como para usuários de SIG em geral, mais uma opção de ferramenta capaz de realizar trabalhos normalmente maçantes. Em relação ao desempenho na tarefa proposta, há espaço para aprimoramento. Dada as circunstâncias certas de treinamento, o modelo pode vir a apresentar resultados mais confiáveis e refinados, enquanto que a estrutura do *plugin* foi desenvolvida de tal forma que permita a fácil manipulação dos pesos empregados no modelo: simplesmente substituindo um arquivo por outro (mais especificamente aquele de extensão ".pth").

Em relação ao desempenho dos métodos clássicos MAXVER e Bhattacharya comparado ao do modelo na detecção de ocupações antrópicas tem-se circunstâncias em que cada técnica demonstra vantagens e desvantagens, principalmente entre Bhattacharya segundo classificação

por região e o modelo, podendo, portanto, ser uma ferramenta importante na confecção de mapas de uso e cobertura do solo. Devido a mudança de paradigma entre classificação e reconhecimento, discutido no item 3.1, o modelo também permite fácil extração dos dados de área e perímetro.

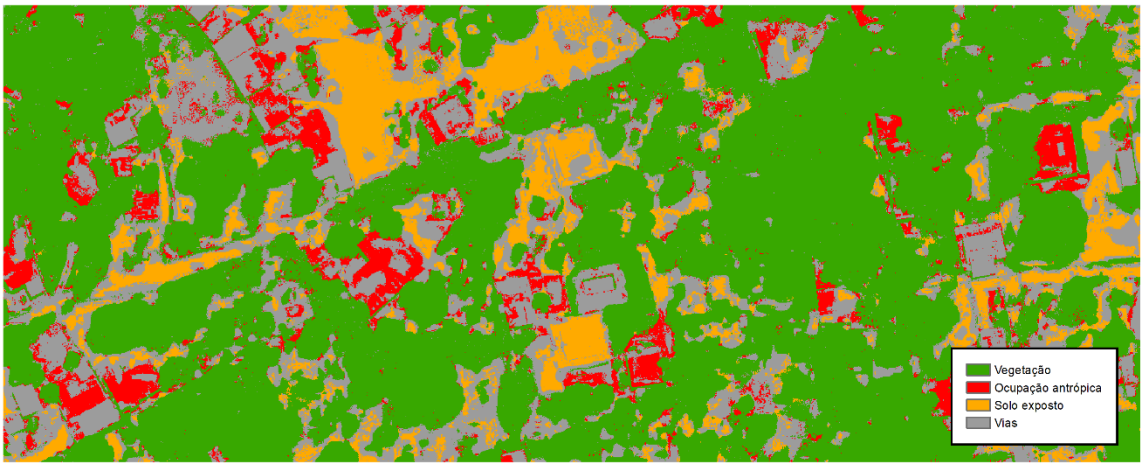
Além da precisão do modelo, outro potencial é a diversificação dos alvos. Este trabalho tem como objetivo ocupações antrópicas, entretanto é completamente viável abarcar veículos, vias e outros alvos na gama de objetos a se detectar. Em relação ao código o qual instancia o modelo basta adaptá-lo e alocar pesos condizentes com o objetivo, preservando a estrutura geral não só do código como do *plugin*.

A interface simples do *plugin* alcança o intuito de manter o extenso processamento e complexidade do modelo distante do usuário, ao mesmo tempo sem limitar o desempenho. No entanto, é importante salientar as limitações do presente trabalho, as quais consistem na imprecisão do modelo, o qual em certas instâncias não detectou os alvos ou os delimitou de forma incorreta. Dessa forma, melhorias podem ser de grande valia para implementação da experiência geral de utilização da ferramenta, assim como conceder flexibilidade ao usuário em relação ao funcionamento dela, tais como a possibilidade de delimitação da região do raster na qual se deseja repassar ao modelo, barra de progresso do processamento e definição do *threshold* não só para detecções como para pixels das segmentações. O conjunto de amostras abarcando a região do DF apontada pela Figura 15 e os pesos treinados estão publicamente disponíveis através do endereço “https://drive.google.com/drive/folders/1r2Lncz-sIW_MIuYGiyvUS_joonwyOX4X?usp=sharing”, assim como o plugin e todos os códigos inerentes ao modelo podem ser acessados através do endereço “https://github.com/AndreEstevam/qgis-mask-rcnn-building-detection”, onde contém um guia de instalação da aplicação no ambiente computacional QGIS.

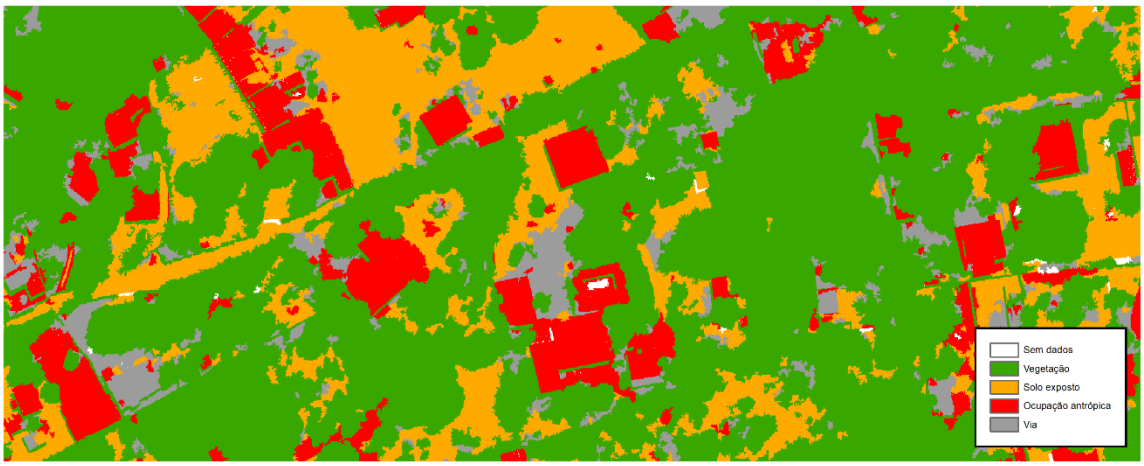
7. APÊNDICE

7.1 Mapas de uso e cobertura do solo

7.1.1 MAXVER



7.1.2 Bhattacharya



7.2 Código

7.2.1 Conjunto de dados

```
class EdfDataset(torch.utils.data.Dataset):
    def __init__(self, dataset_dir, transforms):
        img_dir = os.path.join(dataset_dir, 'image')
        label_dir = os.path.join(dataset_dir, 'label')
        self.img_paths = [os.path.join(img_dir, img) for img in sorted(os.listdir(img_dir))]
        self.mask_paths = [os.path.join(label_dir, mask) for mask in sorted(os.listdir(label_dir))]
        self.transforms = transforms



    def __getitem__(self, idx):
        '''
        Args:
            idx: índice da amostra a ser recuperada;
        return:
            Dicionário contendo:
            - img[PIL.Image]
            - target[dict]:
                - boxes:    FloatTensor[N, 4], N being the n° of instances and it's bounding
                boxe coordinates in [x0, y0, x1, y1] format, ranging from 0 to W and 0 to H;
                - labels:   Int64Tensor[N], class label (0 is background);
                - image_id: Int64Tensor[1], unique id for each image;
                - area:     Tensor[N], area of bbox;
                - iscrowd:  UInt8Tensor[N], True or False;
                - masks:    UInt8Tensor[N, H, W], segmantation maps;
        '''
        img = Image.open(self.img_paths[idx]).convert("RGB")

        mask = load_image_as_np_array(self.mask_paths[idx])
        mask_ch_first = extract_masks_from_cluster(mask, bool_array=True, ch_first=True)
        mask_ch_last = extract_masks_from_cluster(mask, bool_array=True, ch_first=False)
        bboxes = extract_bboxes_from_mask(mask_ch_last)
        num_instaces = np.shape(mask_ch_last)[-1]

        boxes = torch.as_tensor(bboxes, dtype=torch.float32)
        labels = torch.ones((num_instaces,), dtype=torch.int64)
        masks = torch.as_tensor(mask_ch_first, dtype=torch.uint8)
        image_id = torch.tensor([idx])
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])
        iscrowd = torch.zeros((num_instaces,), dtype=torch.int64)

        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
        target["masks"] = masks
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd

        if self.transforms is not None:
            img, target = self.transforms(img, target)

        return img, target



    def __len__(self):
        return len(self.img_paths)


    def check(self):
        for i, j in zip(self.img_paths, self.mask_paths):
            if i[-9:-4] != j[-9:-4]:
                print("Image and label do not match!\n"+i+" - "+j)
```

```
class CrowdDataset(torch.utils.data.Dataset):
    def __init__(self, dataset_dir, subset, transforms, load_small=False):
        dataset_path = os.path.join(dataset_dir, subset)
        if load_small:
            ann_file = os.path.join(dataset_path, "annotation-small.json")
        else:
            ann_file = os.path.join(dataset_path, "annotation.json")
        self.imgs_dir = os.path.join(dataset_path, "images")
        self.coco = COCO(ann_file)
        self.img_ids = sorted(self.coco.getImgIds())
        self.transforms = transforms


    def __getitem__(self, idx):
        '''
        Args:
            idx: índice da amostra a ser recuperada;
        return:
            dict:
                - PIL Image of shape (H, W)
                - target (dict) containing:
                    - boxes:    FloatTensor[N, 4], N being the n° of instances and it's bounding
                    boxe coordinates in [x0, y0, x1, y1] format, ranging from 0 to W and 0 to H;
                    - labels:   Int64Tensor[N], class label (0 is background);
                    - image_id: Int64Tensor[1], unique id for each image;
                    - area:     Tensor[N], area of bbox;
                    - iscrowd:  UInt8Tensor[N], True or False;
                    - masks:    UInt8Tensor[N, H, W], segmantation maps;

                    Image size HARD CODED to 300x300;
        '''
        # Selecting sample
        image_id = self.img_ids[idx]

        # Getting image
        img_obj = self.coco.loadImgs(image_id)[0]
        image = Image.open(os.path.join(self.imgs_dir, img_obj['file_name']))

        # Getting annotations
        anno = self.coco.loadAnns(ids=self.coco.getAnnIds(image_id))
        anno = [obj for obj in anno if obj['iscrowd'] == 0]

        # Getting labels
        classes = torch.ones(len(anno), dtype=torch.int64)

        # Getting masks
        masks_np = np.array([self.coco.annToMask(obj) for obj in anno])
        masks = torch.as_tensor(masks_np, dtype=torch.uint8)

        # Extracting bboxes
        boxes = extract_bboxes_from_mask(np.dstack(masks_np))
        boxes = torch.as_tensor(boxes, dtype=torch.float32)

        # Selecting valid instances
        keep = (boxes[:, 3] > boxes[:, 1]) & (boxes[:, 2] > boxes[:, 0])
        boxes = boxes[keep]
        classes = classes[keep]
        masks = masks[keep]


        image_id = torch.tensor([image_id])
        area = torch.tensor([obj["area"] for obj in anno])
        iscrowd = torch.tensor([obj["iscrowd"] for obj in anno])

        target = {}
        target["boxes"] = boxes
        target["labels"] = classes
        target["masks"] = masks
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd

        if self.transforms is not None:
            image, target = self.transforms(image, target)

        return image, target
```

```
def instantiate_dataloader():
    # Dataset
    dataset = CrowdDataset(DATASET_DIR,
                        subset="train",
                        transforms=get_transform())
    dataset_test = CrowdDataset(DATASET_DIR,
                                subset="val",
                                transforms=get_transform(),
                                load_small=True)

    # Dataloader
    data_loader = torch.utils.data.DataLoader(dataset,
                                            shuffle=True,
                                            num_workers=2,        # wtf is a worker?
                                            collate_fn=collate_fn)
    data_loader_test = torch.utils.data.DataLoader(dataset_test,
                                                shuffle=False,
                                                collate_fn=collate_fn)

    return data_loader, data_loader_test
```

7.2.2 Modelo

```
def get_instance_model(backbone, trainable_layers=0):
    """
    Instancia um modelo de arquitetura mask r-cnn com o dado backbone para a
    detecção de apenas uma classe de objeto.
    Args:
        - backbone[str]: "resnet50+fpn" ou "resnet101+fpn";
        - trainable_layers[int]: camadas do backbone trainàveis, sendo entre 0
        e 5, onde 5 significa todas as camadas treináveis e 0 apenas a "cabeça";
    Returns:
        - model[pytorch.model]: uma instância do modelo de arquitetura mask-rcnn;
    """
    if backbone=="resnet50+fpn":
        model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True,

trainable_backbone_layers=trainable_layers,
                                                                   min_size=300,
                                                                   max_size=300)

        # Atualziando Anchor Generator para a RPN
        model.rpn.anchor_generator = AnchorGenerator(sizes=((8, 16, 32, 64, 128)), aspect_ratios=((1,
1.5, 2)))

        # Trocando preditores(cls+bbox and mask)
        in_features = model.roi_heads.box_predictor.cls_score.in_features
        model.roi_heads.box_predictor = FastRCNNPredictor(in_channels=in_features, num_classes=2)
        in_features_mask = model.roi_heads.mask_predictor.conv5_mask.in_channels
        model.roi_heads.mask_predictor = MaskRCNNPredictor(in_channels=in_features_mask,
dim_reduced=256, num_classes=2)

    elif backbone=="resnet101+fpn":
        # Atualziando Anchor Generator para a RPN
        anchor_gen = AnchorGenerator(sizes=((8, 16, 32, 64, 128)), aspect_ratios=((1, 1.5, 2)))

        # Definindo preditores(cls+bbox and mask)
        box_predictor = FastRCNNPredictor(in_channels=1024, num_classes=2)
        mask_predictor = MaskRCNNPredictor(in_channels=256, dim_reduced=256, num_classes=2)

        # Selecionando backbone
        backbone = BU.resnet_fpn_backbone('resnet101',
                                        pretrained=True,
                                        trainable_layers=trainable_layers)

        model = MaskRCNN(backbone=backbone,
                         min_size=300,
                         max_size=300,
                         rpn_anchor_generator=anchor_gen,
                         box_predictor=box_predictor,
                         mask_predictor=mask_predictor)

    return model
```

7.2.3 Treinamento, validação e teste

```
'''
TREINAMENTO E VALIDAÇÃO
'''
def train(model, optimizer="SGD", lr=0.005, num_epochs=10):
    '''
    Treina um modelo e recupera seus pesos;
    Args:
        - optimizer[string]: otimizador a ser usado; strings aceitos: "SGD" or
        "Adam";
        - model[torch.model]: instância de modelo;
        - lr[float]: taxa de treinamento a ser utilizado;
        - num_epochs[int]: número de épocas
    '''
    params = [p for p in model.parameters() if p.requires_grad]

    assert optimizer=="SGD" or optimizer=="Adam", "Otimizador inválido."

    if optimizer=="SGD":
        optimizer = torch.optim.SGD(params,
                                    lr=lr,
                                    momentum=0.9,
                                    weight_decay=0.001)
    elif optimizer=="Adam":
        optimizer = torch.optim.Adam(params,
                                     lr=lr,
                                     betas=(0.9, 0.999),
                                     eps=1e-08,
                                     weight_decay=0,
                                     amsgrad=False)

    lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                                   step_size=3,
                                                   gamma=0.1)

    for epoch in range(num_epochs):
            E.train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=10000)
            lr_scheduler.step()     # update the learning rate
            E.evaluate(model, data_loader_test, device)
            torch.save(model.state_dict(), "/content/logs/50_state_dict_"+str(epoch)+".pth")

# COnfigurando dispositivo a ser utilizado
device = torch.device('cuda')

# Carregando dados
data_loader, data_loader_test = instantiate_dataloader()

# Carregando modelo
model = get_instance_model(backbone="resnet50+fpn", trainable_layers=5)
model.load_state_dict(torch.load(PRETRAINED_WEIGHTS_PATH))
model.to(device)

# Treinando
train(model, lr=0.00001, num_epochs=2)

'''
TESTE
'''
# Carregando dataset de teste
_, data_loader_test = instantiate_dataloader()

# Carregando modelo treinado
model = get_instance_model(backbone="resnet50+fpn")
model.load_state_dict(torch.load(TRAINED_WEIGHTS_PATH))
model.to(device)

E.evaluate(model, data_loader_test, device)
```

7.3 Métricas de precisão do modelo

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.536
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.847
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.610
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.284
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.668
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.685
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.088
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.510
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.612
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.418
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.729
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.784
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.522
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.847
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.599
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.269
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.649
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.679
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.085
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.493
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.599
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.423
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.707
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.745
```

8. REFERÊNCIAS

AGGARWAL, C. C. **Neural Networks and Deep Learning**. [s.l: s.n.].

BATISTA, J. (2019). **Uso de Redes Neurais Convolucionais na Segmentação de Vias Urbanas Asfaltadas em Imagens de Satélite RGB: estudo de caso em São Luís-MA**. [s.l: s.n.]. Disponível em: <https://sis.sig.uema.br/sigaa/public/programa/noticias_desc.jsf?lc=lc=en_US&id=933&noticia=9717289>. Acesso em: 2 out. 2021.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. [s.l: s.n.]. v. 1

BURGER, W.; BURGE, M. J. **Digital Image Processing**. 79. ed. [s.l: s.n.]. v. 79

CAO, C.; DRAGIĆEVIĆ, S.; LI, S. (2019). **Land-use change detection with convolutional neural network methodsEnvironments - MDPI**. [s.l.]. Disponível em: <https://www.mdpi.com/2076-3298/6/2/25>. Acesso em: 1 set. 2021.

CARVALHO, O. L. F. *et al.* (2021). **Instance segmentation for large, multi-channel remote sensing imagery using mask-RCNN and a mosaicking approach**. [s.l.]. Disponível em: <https://www.mdpi.com/2072-4292/13/1/39>. Acesso em: 18 ago. 2021.

CASTELLUCCIO, M. *et al.* (2015). **Land Use Classification in Remote Sensing Images by Convolutional Neural Networks**. [s.l: s.n.]. Disponível em: <http://arxiv.org/abs/1508.00092>. Acesso em: 1 set. 2021.

CHENG, G.; HAN, J. (2016). **A Survey on Object Detection in Optical Remote Sensing Images**. [s.l: s.n.]. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0924271616300144>. Acesso em: 19 out. 2021.

COURTRAI, L.; PHAM, M. T.; LEFÈVRE, S. (2020). **Small object detection in remote sensing images based on super-resolution with auxiliary generative adversarial networksRemote Sensing**. [s.l.]. Disponível em: <https://www.mdpi.com/2072-4292/12/19/3152>. Acesso em: 1 set. 2021.

CROSTA, A. P. (1999). **Processamento digital de imagens de sensoriamento remoto.** [s.l.] UNICAMP/Instituto de Geociências.

GIRSHICK, R. *et al.* (2014). **Rich feature hierarchies for accurate object detection and semantic segmentation**. [s.l: s.n.]. Disponível em: <http://arxiv.org/abs/1311.2524>. Acesso em: 18 ago. 2021a.

HE, K. *et al.* (2015). **Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition**. [s.l: s.n.]. Disponível em: <http://arxiv.org/abs/1406.4729>.

HE, K. *et al.* (2017). **Mask R-CNN**. Disponível em: < https://arxiv.org/abs/1703.06870>.

Histórico do PDOT – Secretaria de Estado de Desenvolvimento Urbano e Habitação. Disponível em: <http://www.seduh.df.gov.br/historico-do-pdot/>. Acesso em: 6 jun. 2021.

LI, K. *et al.* (2019). **Object Detection in Optical Remote Sensing Images: A Survey and A New Benchmark**. [s.l: s.n.]. Disponível em: <https://arxiv.org/abs/1909.00133>. Acesso em: 19 out. 2021.

LI, S. *et al.* (2019). **3D Virtual Urban Scene Reconstruction from a Single Optical Remote Sensing ImageIEEE Access**. [s.l.] Institute of Electrical and Electronics Engineers Inc., 2019. Disponível em: <https://ieeexplore.ieee.org/document/8710253>. Acesso em: 10 out. 2021.

LIU, W. *et al.* (2016). **SSD: Single Shot MultiBox.** Computer Vision – ECCV 2016. Cham: Springer International Publishing. Disponível em: <https://arxiv.org/abs/1512.02325>. Acesso em: 30 set. 2021.

LONG, J.; SHELHAMER, E.; DARRELL, T. (2014). **Fully Convolutional Networks for Semantic Segmentation**. [s.l: s.n.]. Disponível em: <http://arxiv.org/abs/1411.4038>. Acesso em: 30 set. 2021.

MESQUITA, F. N.; SILVESTRE, K. S.; STEINKE, V. A. **Urbanização e degradação ambiental: Análise da ocupação irregular em áreas de proteção permanente na região administrativa de Vicente Pires, DF, utilizando imagens aéreas do ano de 2016Revista Brasileira de Geografia Física v**. [s.l: s.n.]. Disponível em: <www.ufpe.br/rbgfe>. Acesso em: 13 ago. 2021.

MOHANTY, S. P. *et al.* (2020). **Deep Learning for Understanding Satellite Imagery: An Experimental SurveyFrontiers in Artificial Intelligence**. [s.l.] Frontiers Media SA, 16 nov. Disponível em: <https://www.frontiersin.org/articles/10.3389/frai.2020.534696/full>. Acesso em: 3 set. 2021.

PRITT, M.; CHERN, G. (2020). **Satellite Image Classification with Deep Learning**. [s.l: s.n.]. Disponível em: <https://arxiv.org/abs/2010.06497>. Acesso em: 16 ago. 2021.

REDMON, J. *et al.* (2015). **You Only Look Once: Unified, Real-Time Object Detection**. [s.l: s.n.]. Disponível em: <http://arxiv.org/abs/1506.02640>. Acesso em: 30 set. 2021.

RICHARDS, J. A.; JIA, X. (2006). **Remote sensing digital image analysis: An introduction**. [s.l.] Springer Berlin Heidelberg.

SERMANET, P. *et al.* (2014). **OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks**. [s.l: s.n.]. Disponível em: <http://arxiv.org/abs/1312.6229>. Acesso em: 30 set. 2021.

SKANSI, S. **Introduction to Deep Learning**. Cham: Springer International Publishing, 2018.

SZELISKI, R. **Computer Vision**. 2. ed. [s.l: s.n.]. v. 2

ZHAO, K. *et al.* (2017). **Building Extraction from Satellite Images Using Mask R-CNN with Building Boundary Regularization**. [s.l: s.n.]. Disponível em: <http://arxiv.org/abs/1709.05932>. Acesso em: 28 ago. 2021.

ZHAO, Z.-Q. *et al.* (2019). **Object Detection with Deep Learning: A Review**. [s.l: s.n.]. Disponível em: <http://arxiv.org/abs/1807.05511>. Acesso em: 16 set. 2021.

ZHU, X. X. *et al.* (2017). **Deep learning in remote sensing: a review**. [s.l: s.n.]. Disponível em: <http://arxiv.org/abs/1710.03959>. Acesso em: 20 ago. 2021.