DANIEL GUILLERMO DE ALMEIDA CUETO

CRIAÇÃO DE UMA REDE NEURAL ARTIFICIAL PARA CLASSIFICAÇÃO DO USO E COBERTURA DO SOLO EM IMAGENS DE SATÉLITE

LAVRAS-MG
2021

DANIEL GUILLERMO DE ALMEIDA CUETO

CRIAÇÃO DE UMA REDE NEURAL ARTIFICIAL PARA CLASSIFICAÇÃO DO USO E COBERTURA DO SOLO EM IMAGENS DE SATÉLITE

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia Florestal, para a obtenção do título de Bacharel.

Prof. Dr. Fausto Weimar Acerbi Júnior
Orientador
Vinícius Farnesi Borriello
Coorientador

LAVRAS-MG
2021

DANIEL GUILLERMO DE ALMEIDA CUETO

CRIAÇÃO DE UMA REDE NEURAL ARTIFICIAL PARA CLASSIFICAÇÃO DO USO E COBERTURA DO SOLO EM IMAGENS DE SATÉLITE

CREATION OF AN ARTIFICIAL NEURAL NETWORK FOR LAND USE AND LAND COVER CLASSIFICATION ON SATELLITE IMAGES

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia Florestal, para a obtenção do título de Bacharel.

Aprovada em 04 de maio de 2021
Prof. Dr. Fausto Weimar Acerbi Júnior UFLA
Dr. Eduane José de Pádua UFLA
Ma. Juliana Maria Ferreira De Souza Diniz INPE

Prof. Dr. Fausto Weimar Acerbi Júnior
Orientador
Vinícius Farnesi Borriello
Coorientador

LAVRAS-MG
2021

RESUMO

As Redes Neurais Artificiais são algoritmos criados com o intuito de simular computacionalmente habilidades humanas como aprendizado, generalização, abstração e associação com o objetivo de criar modelos para a resolução de problemas complexos como reconhecimento de voz, classificação de imagens, robótica, visão computacional dentre muitos outros. O presente trabalho buscou expor o processo de criação e implementação de uma Rede Neural Artificial Profunda para a classificação da cobertura do solo em imagens de satélite usando as plataformas *Google Earth Engine, Google Colab e TensorFlow.* Foram geradas 8 distintas classificações da área de estudo para análise da acurácia do algoritmo. O modelo 1 classificou a área em 3 classes distintas: água, vegetação e solo exposto+áreas urbanas e teve uma acurácia global de 93,1%. O modelo 2 classificou a área em 4 classes distintas: água, vegetação nativa, agricultura+pastagens e solo exposto+áreas urbanas e apresentou uma acurácia global de 86,6%. O modelo 3 realizou a classificação da área em 5 classes distintas: água, vegetação nativa, agricultura+pastagens, solo exposto e áreas urbanas e apresentou uma acurácia global de 76,4%. O modelo 4 classificou a área em 6 classes: água, vegetação nativa, solo exposto, agricultura, áreas urbanas e pastagens e apresentou uma acurácia global de 72,6%. Por fim, quatro últimas classificações comparativas foram geradas utilizando uma base de dados de entrada reduzida para analisar o efeito da complexidade e quantidade dos dados de entrada na acurácia da Rede criada. Os modelos comparativos foram executados com os mesmos pontos de treinamento e suas matrizes de confusão geradas com as mesmas amostras de acurácia dos modelos originais e apresentaram acurácia global de 88,4%, 79,0%, 71,1% e 65,2% em comparação aos modelos 1, 2, 3 e 4 respectivamente, representando uma queda média de 7,67% na acurácia global dos modelos com base de dados de entrada reduzida. Com a análise dos resultados observou-se que a Rede Neural Artificial criada apresentou ótima precisão na distinção de 3 classes que decaiu à medida que mais classes foram adicionadas. Ademais, constatou-se que a complexidade e quantidade dos dados de entrada da Rede são de fundamental importância na qualidade das classificações geradas.

Palavras-chave: Sensoriamento remoto. Aprendizado de máquina. *TensorFlow*. *Google Colab*. *Google Earth Engine.*

ABSTRACT

Artificial Neural Networks are algorithms created with the intention of computationally simulating human skills such as learning, generalization, abstraction and association, in order to create models for the solution of complex problems such as speech recognition, image classification, robotics, computer vision among many others. The present work sought to expose in detail the process of creation and implementation of a Deep Artificial Neural Network used for land cover classification in satellite images using the Google Earth Engine, Google Colab and TensorFlow platforms. Eight different classifications of the study area were generated to analyze the accuracy of the algorithm. Model 1 classified the area into 3 distinct classes: water, vegetation and exposed soil + urban areas and had an overall accuracy of 93.1%. Model 2 classified the area into 4 distinct classes: water, native vegetation, agriculture + pasture and exposed soil + urban areas and presented an overall accuracy of 86.6%. Model 3 classified the area into 5 distinct classes: water, native vegetation, agriculture + pastures, exposed soil and urban areas and presented an overall accuracy of 76.4%. Model 4 classified the area into 6 classes: water, native vegetation, exposed soil, agriculture, urban areas and pastures and presented an overall accuracy of 72.6%. Finally, four last comparative classifications were generated using a reduced input database to analyze the effect of the complexity and quantity of the input data on the accuracy of the created Network. The comparative models were performed with the same training points and their confusion matrices generated with the same accuracy samples as the original models and presented an overall accuracy of 88.4%, 79.0%, 71.1% and 65.2% compared to models 1, 2, 3 and 4 respectively, representing an average drop of 7.67% in the overall accuracy of models with reduced input database. With the analysis of the results, it was observed that the Artificial Neural Network created showed excellent precision in the distinction of 3 classes, which declined as more classes were added. In addition, it was found that the complexity and quantity of the input data of the Network are of fundamental importance in the quality of the generated classifications.

Keywords: Remote sensing. Machine learning. TensorFlow. Google Colab. Google Earth Engine.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O ritmo das mudanças climáticas que vêm ocorrendo em nosso planeta nas últimas décadas (BRECHIN; BANDARI, 2011) evidencia a crescente necessidade da modernização de técnicas de monitoramento da superfície terrestre em larga escala (YANG *et al.*, 2013). Para ajudar neste propósito, bases de dados de sensoriamento remoto na proporção dos petabytes (1000 *terabytes)* começaram a ser disponibilizadas gratuitamente por múltiplas agências do governo norte-americano, incluindo a *National Aeronautics and Space Administration* (NASA), a *U.S. Geological Survey* e a *National Oceanic and Atmospheric Administration* (NOAA), além da *European Space Agency* (ESA). O aumento no volume e variedade de dados de sensoriamento remoto disponíveis trouxe dificuldades ao gerenciar, processar e analisar o que é conhecido como *big data*. Segundo Chi *et al*. (2016), este problema é um dilema: Por um lado, temos uma enorme riqueza de *big data* que pode trazer grandes novas oportunidades. Por outro lado, ainda não sabemos como aproveitar essa imensa quantidade de dados com enorme complexidade, diversidade e heterogeneidade e isso faz com que estes dados sejam muito difíceis de processar e analisar em um tempo razoável.

Sabendo da demanda por novas abordagens para manipular e processar estes dados, a empresa americana *Google* lançou a plataforma *Google Earth Engine* (GEE). O GEE é uma plataforma baseada na nuvem que permite a manipulação, processamento e visualização de conjuntos de dados geoespaciais de grandes dimensões de forma rápida e gratuita. O resultado disso, segundo Mutanga e Kumar (2019), é que agora cientistas, pesquisadores independentes, amadores e nações podem explorar esta base de dados para detecção de mudanças, mapeamento de tendências e quantificação de recursos na superfície da Terra de maneiras nunca antes feitas. Diversos estudos já foram publicados analisando a utilidade da plataforma, como o de Robinson *et al.* (2017), que aplicaram o índice de vegetação por diferença normalizada (NDVI) para todo o território dos Estados Unidos utilizando uma coleção de 30 anos de imagens Landsat para compor o resultado. Um outro estudo, realizado por Mahdianpari *et al.* (2019) utilizou o poder de processamento em nuvem da plataforma para manipular dados de alta resolução dos satélites Sentinel 1 e 2 usando algoritmos de aprendizado de máquina e produzir o primeiro mapa detalhado das áreas de pantanal na província de Newfoundland, Canadá. Com o crescimento da popularidade da plataforma, novos estudos e pesquisas estão sendo desenvolvidos trazendo maneiras inovadoras para se manipular, analisar e processar grandes bases de dados de sensoriamento remoto.

O presente trabalho tem como objetivo demonstrar o processo de implementação e analisar o potencial de uma destas novas maneiras de se trabalhar com grandes bases de dados de sensoriamento remoto: as Redes Neurais Artificiais (RNAs). As RNAs são modelos matemáticos computacionais inspirados nos princípios de funcionamento dos neurônios e na estrutura do cérebro animal criados com o intuito de simular computacionalmente habilidades como aprendizado, generalização, abstração e associação de conhecimento através da experiência. Estes modelos são usados na resolução de diversos problemas como a extração de padrões complexos de volumes massivos de dados, indexação semântica, rotulação de dados, rápida extração de informação e simplificação de tarefas discriminativas (NAJAFABADI *et al.* 2015) e são especialmente indicados para aplicações que envolvem grande quantidade e/ou complexidade de dados de entrada.

1.1 Objetivos

1.1.1 Objetivo Geral

Demonstrar o processo de criação e implementação de uma rede neural artificial profunda utilizando a plataforma *Google Earth Engine* em conjunto com a biblioteca *TensorFlow* para realizar a classificação da cobertura e uso do solo em imagens de satélite.

1.1.2 Objetivos específicos

Analisar a acurácia da Rede Neural Artificial criada para classificação de imagens com números de classes diferentes por meio de análise visual, matriz de confusão e coeficiente Kappa.

Analisar o efeito da complexidade e tamanho da base de dados de entrada no algoritmo criado.

Demonstrar o potencial das plataformas apresentadas para a realização de trabalhos, estudos e pesquisas que envolvam grandes bases de dados de sensoriamento remoto, também chamadas por autores de *big data*.

2 REFERENCIAL TEÓRICO

2.1 Sensoriamento Remoto

Nas últimas décadas o avanço das tecnologias geoespaciais disponíveis fez com que o uso do sensoriamento remoto para o monitoramento da superfície terrestre se tornasse uma prática comum. Segundo Jensen e Epiphanio (2009), sensoriamento remoto é a arte e a ciência de obter informação sobre um objeto sem estar em contato físico com o mesmo. Gates *et al.* (1965) foi um dos precursores em pesquisa no tema. O autor pontua que parte da radiação solar que incide na Terra interage com objetos, podendo ser transmitida, refletida ou absorvida. Usando um sensor é possível mensurar a parte da radiação que é refletida tornando possível avaliar isoladamente o comportamento de cada objeto, chamado comportamento espectral.

Figueiredo (2005) mostra que a radiação eletromagnética (REM) solar interage de maneira distinta com cada objeto ou feição terrestre, principalmente devido às composições físico-químicas únicas de cada um deles. Em decorrência desta interação, a radiação que é refletida dos alvos leva aos sensores a assinatura espectral dos mesmos, que é definida pela quantidade de energia que será refletida pelo objeto em cada diferente faixa do espectro eletromagnético.

Meneses e Almeida (2012) lembram que apesar da REM ser vista como um espectro contínuo, o espectro eletromagnético foi arbitrariamente dividido pelo homem em intervalos de comprimentos de onda cuja nomenclatura de cada intervalo foi feita baseada no uso que o homem encontrou para sua aplicação. Na Tabela 1, observam-se os intervalos espectrais muitas vezes empregados pelos sensores remotos.

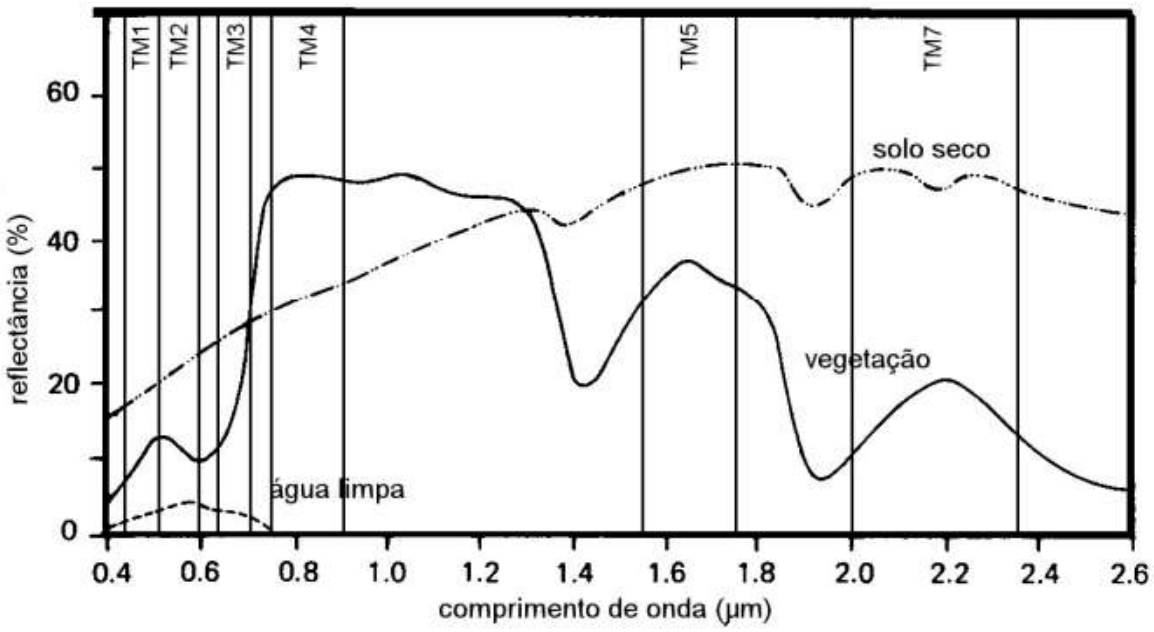
Tabela 1 - Intervalos espectrais possíveis de serem usados pelos sensores remotos.

0,45 – 0,76μm	Visível
0,76 – 1,20μm	Infravermelho próximo
1,20 – 3,00μm	Infravermelho de ondas curtas
3,00 – 5,00μm	Infravermelho médio
5,00 μm – 1.000 μm (1 mm)	Infravermelho termal
1.000 μm – 1.000.000 μm (100 cm)	Micro-ondas

Fonte: Adaptado de Meneses e Almeida (2012).

O intervalo entre 0,45 e 0,76μm é chamado de visível pois é a faixa que é possível ser observada pelo olho humano. Algumas informações, porém, não são possíveis de serem observadas na faixa visível, por isto se faz necessário o uso do espectro infravermelho próximo (NIR), de ondas curtas (SWIR) e médio em algumas situações. A figura 1 mostra a assinatura espectral da água limpa, do solo e da vegetação.

Figura 1-Assinatura espectral da água limpa, solo seco e vegetação.



Fonte: Adaptado de Lillesand, Kiefer e Chipman (2015).

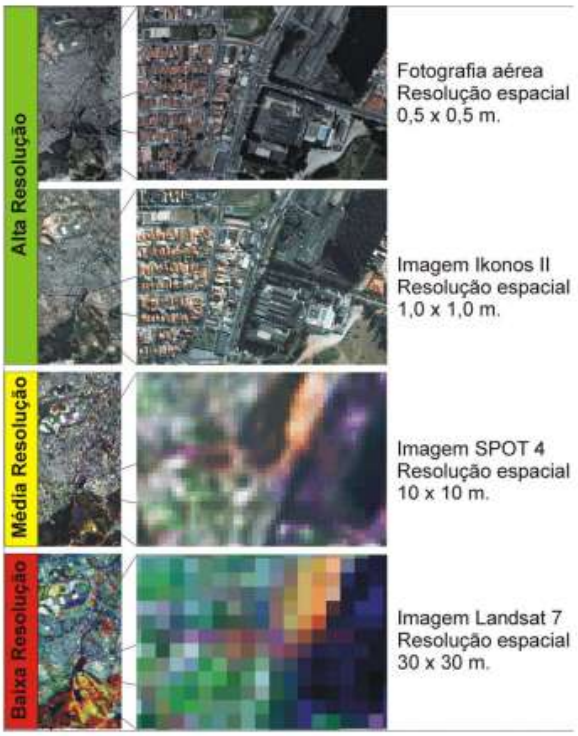
A partir das faixas espectrais coletadas nos sensores é possível formar uma imagem em composição RGB, ou seja, a partir das cores vermelho (*red*), verde (*green*) e azul (*blue*). O espectro RGB proporciona a visualização de imagens entre 0,45 e 0,76 μm de comprimento de onda, faixa visível ao olho nu. Utilizando a combinação de diferentes faixas espectrais representadas na composição RGB é possível destacar alvos na superfície terrestre com maior facilidade. A combinação de faixas utilizada irá depender dos alvos que devem ser identificados e do sensor usado no trabalho. Por exemplo, é imprescindível o uso da faixa de onda entre 0,76 e 1,3 μm, que corresponde à faixa do infravermelho próximo, quando o objetivo é diferenciar feições como a vegetação e o solo. Isso acontece pois cada objeto possui uma interação única com a REM solar e faz com que a assinatura espectral do mesmo, como observado na Figura 1, possua um comportamento muito característico em diferentes faixas do espectro eletromagnético.

2.1.1 Resoluções das imagens de satélite

A detecção ou identificação de um objeto nas imagens de sensoriamento remoto é determinada por quatro diferentes formas de medição: a área do campo de visada do sensor, o comprimento de onda das bandas, os valores numéricos da medida da radiância do alvo e a data em que a imagem é tomada (MENESES; ALMEIDA, 2012). Estas quatro formas de medição definem as resoluções espacial, espectral, radiométrica e temporal respectivamente.

A resolução espacial, segundo Lillesand, Kiefer e Chipman (2015), é uma indicação de quão bem um sensor consegue captar detalhes espaciais. É uma medida da área da superfície terrestre representada por cada pixel em uma imagem. Um sensor com resolução espacial de 30 metros significa que cada pixel da imagem tirada por ele corresponde a um quadrado de 30 por 30 metros na superfície terrestre. A qualidade da resolução espacial é inversamente proporcional à área representada por cada pixel da imagem. Na Figura 2 observa-se o efeito da resolução espacial em uma imagem.

Figura 2-Efeito da resolução espacial em uma imagem.



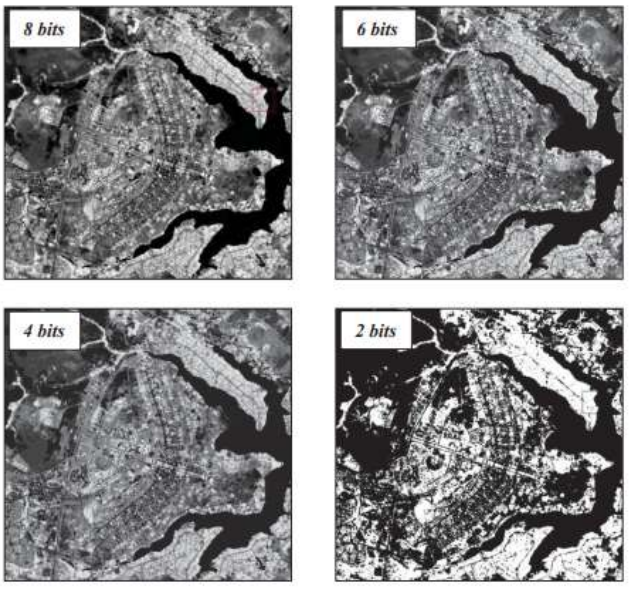
Fonte: Melo (2002).

A resolução espectral é um conceito que se refere à habilidade de um sensor de conseguir distinguir diferentes feições e objetos baseado em suas propriedades espectrais. Para Meneses e Almeida (2012), a resolução espectral envolve três parâmetros de medida: o número de bandas que o sensor possui, a largura em comprimento de onda das bandas e as

posições que as bandas estão situadas no espectro eletromagnético. Sensores considerados de melhor resolução possuem um maior número de bandas situadas em diferentes posições do espectro eletromagnético com larguras estreitas de comprimento de onda, pois possuem maior capacidade de diferenciar e caracterizar alvos na superfície terrestre. Imagens com boa resolução espectral permitem que alvos individualizados (tipos de solo, água, vegetação, plantações, etc.) possam ser caracterizados mais facilmente devido às assinaturas espectrais únicas de cada feição.

A resolução radiométrica se refere à capacidade do sensor em diferenciar as variações da radiância espectral recebida em cada pixel. Expressa em termos de números digitais binários (*bits*), ela corresponde à quantidade de níveis de cinza ou níveis digitais com os quais a imagem é representada. O valor do intervalo é sempre expresso em potência de 2, já que os níveis são em *bits.* Isso significa que uma imagem com 2 *bits* de resolução radiométrica terá $2^2 = 4$ níveis de cinza em sua composição. Uma com 4 bits terá $2^4 = 16$ níveis de cinza e assim por diante. Quanto maior o valor, maior será a qualidade da imagem. A Figura 3 mostra imagens com 4 resoluções radiométricas diferentes. Observa-se que o detalhamento visual da imagem é melhor quanto maior a resolução radiométrica.

Figura 3- Exemplos de imagens com diferentes níveis de quantização ou de resolução radiométrica.



Fonte: Meneses e Almeida (2012).

Por fim, a resolução temporal refere-se à frequência que o sensor revisita uma área e obtém novas imagens. Ela é quantificada pelo intervalo de tempo mínimo entre diferentes obtenções.

> A resolução temporal é fundamental para acompanhar ou detectar a evolução ou mudanças que ocorrem na Terra, principalmente para alvos mais dinâmicos, como o ciclo fenológico de culturas, desmatamentos, desastres ambientais, tendo forte impacto na monitoração ambiental. (MENESES E ALMEIDA, 2012, p. 32).

A Tabela 2 mostra as características de alguns dos sensores que são amplamente usados na área de sensoriamento remoto.

Tabela 2 - Resolução espacial, espectral, radiométrica e temporal dos satélites Landsat 8, Sentinel-2, QuickBird e RapidEye.

Satélite	**Resolução**			
	Espacial	**Espectral**	**Radiométrica**	**Temporal**
Landsat-8/OLI-TIRS	Banda pancromática: 15m Bandas multiespectrais: 30m Bandas termais: 100m	11 bandas	16 *bits*	16 dias
Sentinel-2	Bandas visíveis: 10m Bandas multiespectrais: 20m Bandas auxiliares: 60m	13 bandas	12 *bits*	5 dias
QuickBird	Banda pancromática: 0,65m Bandas multiespectrais: 2.62m	5 bandas	11 *bits*	1 a 3,5 dias
RapidEye	5m	5 bandas	12 *bits*	Diariamente

Fonte: Do autor (2021).

2.1.2 Processamento digital de imagens

O processamento digital de imagens envolve uma série de procedimentos aplicados a imagens visando aprimorar suas qualidades espectrais e espaciais para que sejam mais apropriadas para uma determinada aplicação. Segundo Machado e Quintanilha (2008), "O propósito geral do processamento de imagem é preparar o dado de imagem para que este possa ser melhor utilizado nas etapas subseqüentes de interpretação e extração de informações."

Lillesand, Kiefer e Chipman (2015) categorizaram os processamentos possíveis de serem realizados em sete diferentes amplos tipos de operações assistidas por computadores:

a) Pré-processamento: são operações realizadas com o intuito de corrigir dados distorcidos ou degradados de uma imagem para que ela represente mais precisamente a cena original. Envolve o processamento inicial da imagem bruta para corrigir

distorções geométricas, para calibrar o dado radiometricamente e para eliminar ruídos presentes na imagem;

b) Realces são procedimentos usados para melhorar a distinção visual entre diferentes feições em uma cena, aumentando o número de informações que podem ser interpretadas na imagem. Souza e Correia (2007) mostram a eficácia de diferentes técnicas de realce como a limiarização, a equalização de histograma e algumas técnicas de filtragem espacial;

c) Classificação de imagens: é o processo de extração de informação de imagens para reconhecer os padrões e objetos dessa imagem, relacionando-o com uma classe temática. Normalmente envolve a análise multi ou hiperespectral de uma imagem e a aplicação de regras de decisão baseadas em estatística para determinar a cobertura do solo em cada pixel;

d) Análise das mudanças: São operações realizadas averiguando imagens de dois diferentes pontos no tempo para determinar a natureza e a extensão de modificações ocorridas em uma área. A proliferação de bases de dados de sensoriamento remoto gratuitas cada vez mais extensas e complexas facilita a realização de estudos nesta área;

e) Integração com Sistemas de Informação Geográfica (SIG): são procedimentos onde os dados georreferenciados de uma imagem são cruzados com outras bases de dados para se obter informações mais ricas de determinada região. Por exemplo, dados de imagem são frequentemente combinados com solo, topografia, pluviosidade, temperatura e outros parâmetros para uma análise mais profunda da localidade estudada;

f) Análise de imagens hiperespectrais: Todos os processamentos realizados em imagens multiespectrais descritos até aqui podem ser estendidos a imagens hiperespectrais. As bases de dados hiperespectrais, porém, possuem uma natureza muito mais complexa e volumes muito maiores de dados que as imagens multiespectrais, fazendo-se necessário a criação de procedimentos específicos para análise de imagens de tal natureza. Bioucas-Dias *et al.* (2013) explicita que a dificuldade de se trabalhar com imagens hiperespectrais se encontra na dimensionalidade e dimensão dos dados, a mistura espectral (linear e não-linear) e os mecanismos de degradação associados ao processo de medição, como ruído e efeitos atmosféricos;

g) Modelagem biofísica: O objetivo da modelagem biofísica é relacionar quantitativamente os dados digitais adquiridos por um sistema de sensoriamento

> remoto com características biofísicas e fenômenos passíveis de serem mensurados na superfície terrestre, como o rendimento de uma colheita, concentração de poluição ou determinação da profundidade da água em determinada área. Jensen (1983) demonstra o uso do sensoriamento remoto para aferição de nove diferentes variáveis biofísicas na superfície terrestre: elevação, cor e assinatura espectral de feições, características de absorção da clorofila, biomassa da vegetação, teor de umidade da vegetação, teor de umidade do solo, temperatura da superfície e textura ou rugosidade da superfície.

Os procedimentos de processamento digital de imagens na grande maioria das vezes são utilizados em combinação, como parte de uma sequência de operações a serem feitas com o intuito de transformar os dados coletados pelos sensores em um produto com mais utilidade para a finalidade proposta.

2.2 Aprendizado de máquina

O aprendizado de máquina, também conhecido como *machine learning*, é uma área da inteligência artificial que visa desenvolver algoritmos e técnicas computacionais que possibilitem que computadores sejam capazes de aprender, ou seja, que eles consigam adquirir conhecimento automaticamente a partir de exemplos e usá-lo para aperfeiçoar seu desempenho em dada tarefa (GOLDSCHMIDT, 2010).

Goldschmidt (2010) pontua também que o processo de aprendizado só é possível devido aos sistemas de aprendizado. Segundo Monard e Baranauskas (2003, p. 39), “sistema de aprendizado é um programa de computador que toma decisões baseado em experiências acumuladas através da solução bem sucedida de problemas anteriores.”. Os algoritmos de *machine learning* tomam decisões baseados em exemplos históricos que são representados por um conjunto de características, também chamadas de atributos, que variam em função de sua aplicação.

Os atributos escolhidos como exemplos de um algoritmo devem representar bem o conjunto de dados como um todo, além de possuir a amplitude necessária para ajudar na distinção de classes ou usos dentro da aplicação e possuem enorme influência na qualidade do conhecimento gerado pelo sistema de aprendizado. De acordo com Monard e Baranauskas (2003), os algoritmos partem de atributos específicos e procuram construir modelos que representem não só com os exemplos utilizados no aprendizado, mas também possíveis novos exemplos que possam surgir.

O aprendizado de máquina gera conhecimento e aprende a partir de dados e padrões pelo processo de indução, que é a forma de inferência lógica que permite conclusões a partir de um conjunto particular de exemplos. Neste processo, a qualidade dos atributos escolhidos é de suma importância para que as hipóteses testadas pelo modelo tenham validade. Grandes volumes de amostras de dados são necessários para conseguir aprender e gerar conhecimento relevante para a aplicação em questão e as variáveis nesse conjunto de dados devem trazer valor e gerar o maior número de hipóteses para o algoritmo aprender (CARVALHO *et al.*, 2011).

No aprendizado pelo processo de indução, que é base para os principais algoritmos de *machine learning* da atualidade, é comum que o modelo de conhecimento seja gerado a partir de sucessivas iterações sobre o conjunto de dados disponíveis. Monard e Baranauskas (2003) destacam duas abordagens distintas para o aprendizado de máquina: Aprendizado supervisionado e não-supervisionado.

No aprendizado supervisionado, um conjunto de amostras bem definidas e rotuladas é utilizado como base de treinamento do algoritmo, que cria regras de decisão baseadas nos atributos do conjunto de amostras para estimar o rótulo de novos dados. As amostras são compostas por um ou mais valores de entrada e apenas um valor de saída, que é a representação numérica da classe que aquela amostra possui maior probabilidade de estar inserida.

O aprendizado não-supervisionado não utiliza um conjunto de amostras de treinamento, de maneira que as regras de decisão do algoritmo são desenvolvidas durante sua execução. Batista (2003) pontua que nesta abordagem de aprendizado é fornecido ao sistema apenas um conjunto de exemplos sem incluir a informação sobre a classe de cada um e a partir disso é construído um modelo que busca por regularidades nos exemplos e forma agrupamentos de amostras com características similares. Após a determinação dos agrupamentos, na maioria das vezes faz-se necessário uma análise adicional para determinar o significado de cada grupo no contexto de sua aplicação.

A Figura 4 mostra a hierarquia de aprendizado descrita por Carvalho *et al.* (2011), onde o processo indutivo se encontra no topo, seguido dos sistemas supervisionados, que seriam as tarefas preditivas de classificação e regressão e dos não-supervisionados, representados pelas tarefas descritivas como o agrupamento, associação e sumarização.

Figura 4- Hierarquia de aprendizado.

Fonte: Carvalho *et al.* (2011).

2.2.1 Paradigmas de aprendizado

Um paradigma de aprendizado diz respeito à forma que um algoritmo extrai conhecimento a partir de um conjunto de exemplos. Goldschmidt (2010) descreve os principais paradigmas de aprendizado sobre os quais os algoritmos de *machine learning* se baseiam:

a) Simbólico: Os sistemas de aprendizado simbólico buscam aprender construindo representações simbólicas de um conceito através da análise de exemplos e contra-exemplos desse conceito. O modelo de conhecimento pode ser representado por expressões lógicas, árvores, regras, redes semânticas, etc.
b) Estatístico: Neste paradigma, modelos estatísticos são usados para encontrar uma boa aproximação do conceito induzido. Monard e Baranauskas (2003) pontuam que alguns autores têm considerado Redes Neurais como métodos estatísticos paramétricos, uma vez que treinar uma Rede Neural geralmente significa encontrar valores apropriados para pesos e parâmetros pré-determinados.
c) Baseado em Exemplos: Uma forma de classificar um exemplo é lembrar de outro similar cuja classe é conhecida e assumir que o novo exemplo terá a mesma classe. Este tipo de paradigma classifica exemplos nunca vistos através de exemplos similares conhecidos. Os algoritmos mais conhecidos deste tipo de paradigma são o de Vizinho mais Próximo e Raciocínio Baseado em Casos.
d) Conexionista: Faz o uso de modelos matemáticos simplificados baseados no modelo biológico do sistema nervoso para tentar abstrair mapeamentos de novos exemplos nas saídas desejadas ou agrupamentos de exemplos similares. Os modelos, também

chamados de Redes Neurais, envolvem unidades altamente interconectadas e, por esse motivo, o nome conexionismo é utilizado para descrever este tipo de algoritmo.

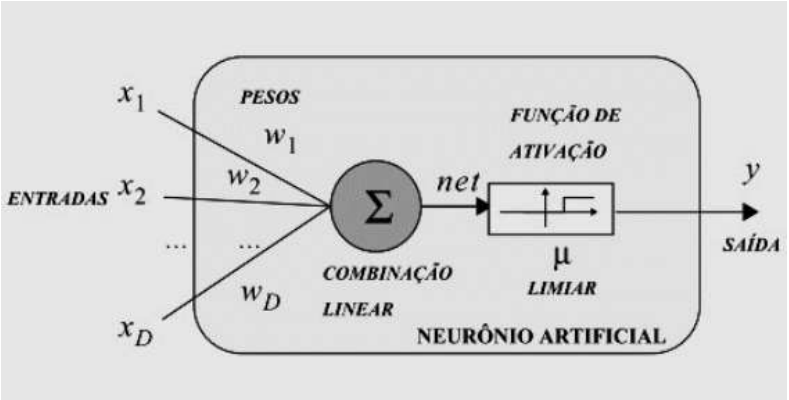
e) Evolucionário ou Genético: Este paradigma baseia-se nos modelos biológicos da evolução natural e reprodução genética onde uma população de elementos de classificação competem entre si para criar um modelo que solucione o problema em questão. Elementos com uma performance fraca são descartados enquanto os elementos mais fortes se proliferam e formam variações de si mesmos (MONARD; BARANAUSKAS, 2003).

2.2.2 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNA), segundo Goldschmidt (2010), são modelos matemáticos que se baseiam nos princípios de funcionamento dos neurônios e na estrutura do cérebro para adquirir, armazenar e utilizar conhecimento experimental com o intuito de simular computacionalmente habilidades humanas como aprendizado, generalização, abstração e associação. Estes modelos são dispostos em uma ou mais camadas que são interligadas por um grande número de conexões ou nós que simulam os neurônios do cérebro humano. Cada nó possui pesos que armazenam o conhecimento representado no modelo e servem para ponderar a entrada recebida por cada neurônio da rede. Uma Rede Neural Artificial é considerada profunda quando contém duas ou mais camadas intermediárias em sua arquitetura.

O primeiro modelo de RNA foi proposto por McCulloch e Pitts (1943) onde os autores buscaram descrever um modelo artificial de um neurônio e apresentar suas capacidades computacionais. Neste modelo, o neurônio artificial possuía apenas uma saída, produzida em função da soma dos valores de suas diversas entradas, como representado na Figura 5.

Figura 5- Modelo de neurônio artificial de McCulloch e Pitts (1943).

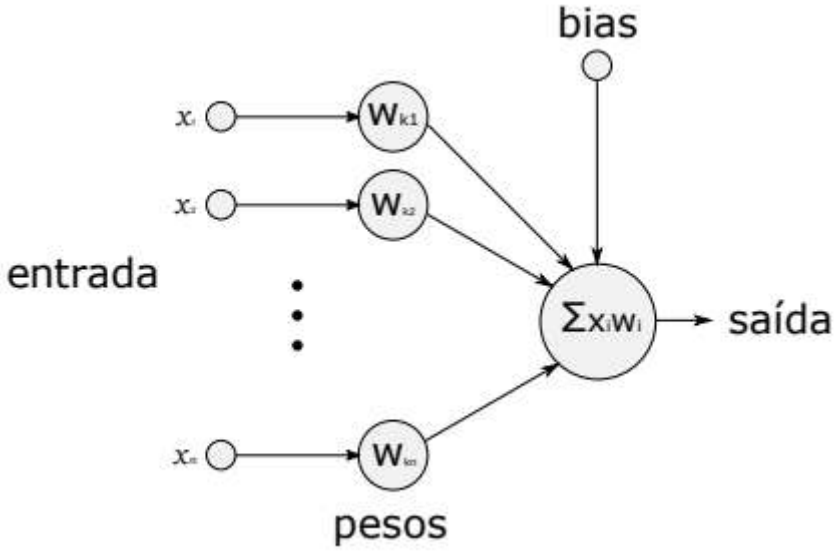


Fonte: Vale *et al.* (2016).

Vale *et al.* (2016) explica que as RNAs podem apresentar diferentes arquiteturas que são definidas pela forma que os neurônios se conectam e as redes formadas. Redes com apenas uma camada de nós como o modelo de McCulloch-Pitts só conseguem resolver problemas linearmente separáveis enquanto redes com mais camadas são mais apropriadas para solucionar problemas que envolvem processamento temporal ou de dados mais complexos.

A partir dos estudos de McCulloch e Pitts foram desenvolvidos novos tipos de modelagem de RNAs, como o modelo de Rosenblatt (1958), conhecido como *perceptron* que consiste de uma camada de entrada conectada por caminhos com pesos fixados ao neurônio que está associado e os pesos dessas conexões são ajustáveis. O modelo de Rosenblatt é demonstrado na Figura 6.

Figura 6 - Modelo de *perceptron* de Rosenblatt (1958).

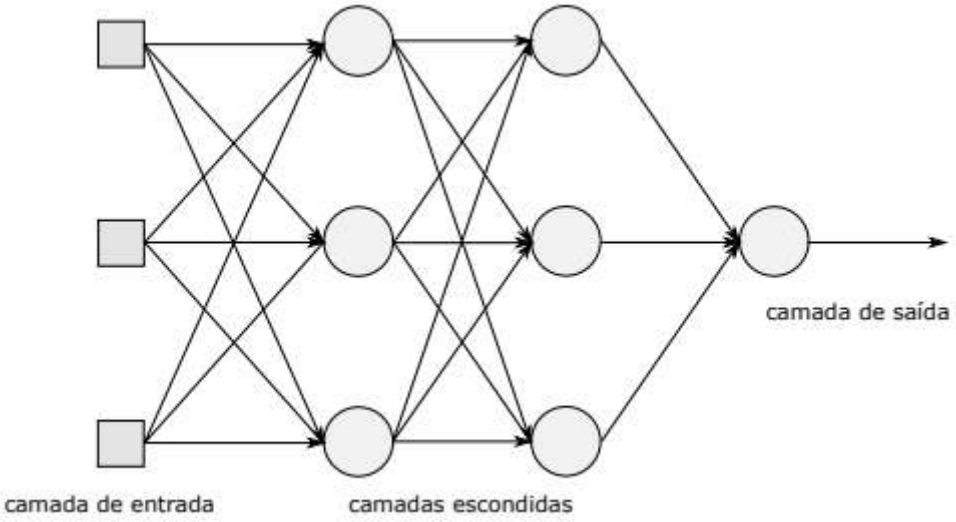


Fonte: Pinto (2018).

Posteriormente, Cybenko (1989) demonstrou que uma rede que possua uma camada intermediária pode implementar qualquer função contínua e a utilização de duas camadas intermediárias permite a aproximação de qualquer função. As redes com uma ou mais camadas intermediárias ou escondidas são chamadas de *MultiLayer Perceptron* (MLP) e possuem um poder computacional maior que as redes de camada única devido ao fato que são capazes de tratar dados que não são linearmente separáveis. A definição da arquitetura de uma rede MLP está relacionada ao número de nós ou neurônios das camadas intermediárias e

como eles se interconectam. Esta definição dependerá da distribuição dos padrões de treinamento e validação da rede. A Figura 7 representa uma MLP comum.

Figura 7 - Rede MLP com duas camadas intermediárias.



Fonte: Pinto (2018).

Goldschmidt (2010) pontua que o algoritmo de treinamento mais usado nas redes MLP é o de retropropagação de erro, também conhecido como *back-propagation*. O *back-propagation* é um algoritmo que utiliza pares de entrada e saída para através de um mecanismo de correção de erros ajustar os pesos da rede. Pinto (2018) explica que o algoritmo ocorre em duas fases que são chamadas de *forward* e *backward*. O algoritmo define para cada padrão de treinamento uma saída na fase *forward* que é comparada com a saída desejada, e os pesos dos nodos são atualizados na fase *backward*. O número de execuções do algoritmo de *back-propagation*, segundo Vale *et al.* (2016), será determinado de acordo com os critérios de parada escolhidos, como, por exemplo, um número definido de N ciclos, encerrar após o erro médio ficar abaixo de uma constante definida ou quando o percentual de classificações corretas ficar acima de certa porcentagem. O autor evidencia também que as redes *perceptron* multicamadas são as RNAs mais utilizadas em diversas aplicações atualmente, devido à sua simplicidade e precisão.

3 MATERIAIS E MÉTODOS

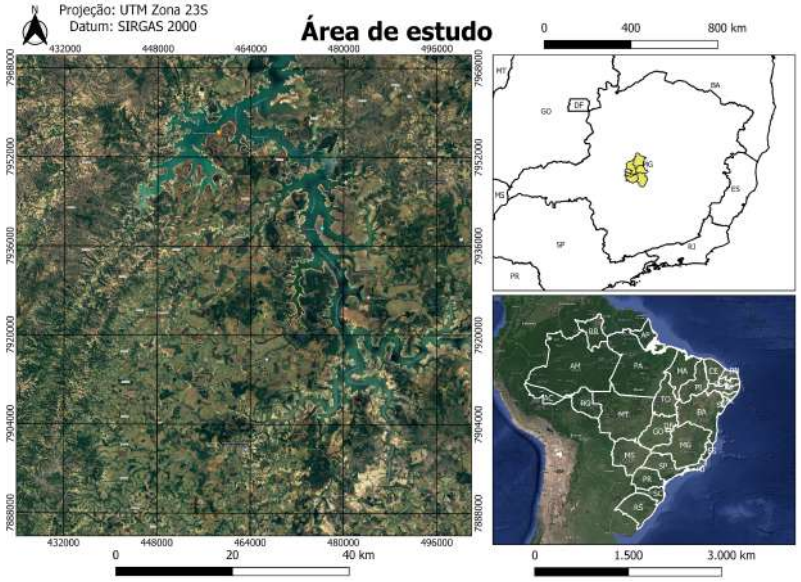
Usando a plataforma GEE em conjunto com o *framework TensorFlow* foi criado um modelo de rede neural artificial profunda para a classificação do uso e cobertura do solo em uma área no estado de Minas Gerais. O *Tensorflow* é um *framework open-source*, também criado pela empresa americana *Google*, usado para auxiliar na implementação e execução de algoritmos de *machine learning*. A ferramenta já tem sido usada para conduzir pesquisas e implementar algoritmos de aprendizado de máquina em diversas áreas como reconhecimento de voz, visão computacional, robótica, recuperação de informações, extração de informações geográficas, entre outras (ABADI *et al.*, 2016).

3.1 Área de estudo

O estudo foi realizado na área da represa de Três Marias no rio São Francisco, no estado de Minas Gerais na região sudeste do Brasil. A classificação foi realizada numa área de 593.586 hectares que compreende os limites dos municípios de Abaeté, Biquinhas, Felixlândia, Morada Nova de Minas, Paineiras, Pompéu e Três Marias.

De acordo com a classificação climática de Köppen (1948), a área tem clima tropical com inverno seco (Aw). Euclydes *et al.* (2001) demonstra que a geologia da região é composta por rochas de idade pré-cambriana, desde arqueanas até proterozóicas superiores, a vegetação é constituída predominantemente pelo bioma Cerrado e a temperatura média anual varia de 19-23ºC. A área de estudo é demonstrada na Figura 8.

Figura 8 - Mapa da região de estudo.



Fonte: Do autor (2021).

3.2 Plataformas e *softwares* utilizados

3.2.1 *Google Earth Engine* (GEE)

Google Earth Engine é uma plataforma baseada na nuvem que possibilita o acesso a recursos de computação de alto desempenho para o processamento de conjuntos de dados geoespaciais extensos sem a necessidade de conhecimento técnico avançado para a manipulação destes dados. Ademais, Gorelick (2017) pontua que a plataforma é projetada para ajudar pesquisadores a disseminar facilmente seus resultados para outros pesquisadores, formuladores de políticas, ONGs, trabalhadores de campo, e até mesmo o público em geral.

A plataforma em si consiste em um catálogo de vários *petabytes* de dados prontos para análise em conjunto com um serviço de computação de alta performance intrinsecamente paralelo. Tudo isto é acessado através de uma interface *web* de programação de aplicativo (API) associada com um ambiente de desenvolvimento interativo (IDE) que possibilita uma rápida prototipagem e visualização dos resultados.

Tamiminia *et al.* (2020) realizaram uma análise investigativa de artigos revisados e publicados usando a plataforma com foco em vários recursos, incluindo dados, tipo de sensor, área de estudo, resolução espacial, aplicação, estratégia e métodos analíticos. Foram analisados 349 artigos de 146 diferentes periódicos do período de 2010 até Outubro de 2019. O estudo serviu para fornecer aos leitores uma visão geral do quadro atual e tendências gerais de uso relacionadas à plataforma GEE.

3.2.2 *TensorFlow*

O *TensorFlow* é uma plataforma de código aberto criada para facilitar a implementação de algoritmos de aprendizado de máquina, computação numérica e outras tarefas. Formado por um ecossistema abrangente e flexível de ferramentas, bibliotecas e recursos da comunidade, permite que pesquisadores trabalhem com algoritmos de *machine learning* de última geração e que desenvolvedores criem e implementem aplicativos usando tecnologia de aprendizado de máquina com facilidade.

Abadi *et al.* (2016) descreve o funcionamento da plataforma, que usa gráficos de fluxo de dados para representar a computação, o estado compartilhado e as operações que alteram esse estado. O maior benefício que *TensorFlow* oferece para o desenvolvimento de algoritmos

de aprendizado de máquina é a abstração. A arquitetura da plataforma permite que pesquisadores e desenvolvedores foquem seus esforços na lógica geral de sua aplicação e não precisem lidar com detalhes técnicos complexos de computação matemática envolvidos na implementação de um algoritmo de *machine learning* como configurar individualmente as conexões entre os nós da rede.

A plataforma conta com a funcionalidade da API Keras, que foi usada no presente trabalho. Keras é uma API de alto nível usada para facilitar o processo de implementação dos algoritmos de *machine learning*. A API permite o ajuste, avaliação e execução de modelos de aprendizado de máquina com apenas algumas linhas de código, possibilitando a implementação de tais algoritmos sem a necessidade de conhecimento técnico avançado em programação e matemática computacional.

3.2.3 *Google Colaboratory*

Google Colaboratory, também comumente chamado de *Google Colab* ou apenas Colab, é um serviço gratuito hospedado na nuvem que permite que qualquer pessoa escreva e execute código em Python usando apenas o navegador, sem a necessidade de nenhum tipo de instalação de software em uma máquina. É especialmente adequado para implementação de algoritmos de aprendizado de máquina, inteligência artificial, análise de dados e educação. O Colab é um serviço de *notebook Jupyter* hospedado na nuvem que não requer configuração para uso, ao mesmo tempo que fornece acesso gratuito a recursos de computação com alto poder de processamento, incluindo *graphic processing units* (GPUs).

O serviço é composto por uma lista de células que podem conter textos, imagens, vídeos e, principalmente, códigos executáveis com suas respectivas saídas. A execução de códigos e algoritmos de alta complexidade na plataforma é toda feita pela nuvem de processamento da empresa *Google*, sendo necessário apenas um navegador com conexão com a internet para a execução de aplicações que requerem grande poder de processamento.

3.3 Metodologia

Para realizar a classificação de uso e cobertura do solo na área de estudo foi utilizado um algoritmo de Rede Neural Artificial Profunda com uma camada de entrada, duas camadas ocultas com 64 nodos cada, uma camada *dropout* e uma camada de saída. A base de dados da camada de entrada foi uma imagem composta das estatísticas das bandas visíveis, do infravermelho próximo e do infravermelho de ondas curtas de todas imagens adquiridas pelo

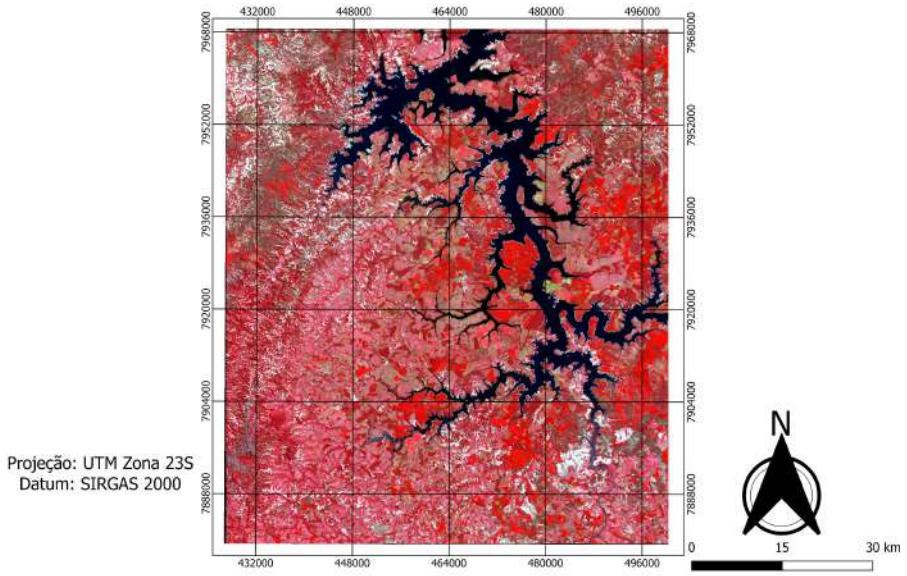
satélite Landsat-8/OLI-TIRS no ano de 2020. As estatísticas usadas foram a média, mediana, desvio padrão, valor máximo e valor mínimo de cada pixel, resultando em 30 variáveis ao final do processo. A tabela 3 apresenta as variáveis relacionadas às estatísticas das bandas espectrais do sensor Landsat-8 e a Figura 9 mostra a imagem final com a composição de bandas em falsa-cor, usando a combinação das bandas medianas 5, 4 e 3. O código fonte utilizado pode ser observado no Apêndice A.

Tabela 3- Variáveis relacionadas às estatísticas das bandas espectrais do Landsat-8 utilizadas como base de dados de entrada.

Bandas	**Média**	**Mediana**	**Desvio Padrão**	**Mínimo**	**Máximo**
Azul (0.452-0.512 µm)	B2_mean	B2_median	B2_stdDev	B2_min	B2_max
Verde (0.533-0.590 µm)	B3_mean	B3_median	B3_stdDev	B3_min	B3_max
Vermelho (0.636-0.673 µm)	B4_mean	B4_median	B4_stdDev	B4_min	B4_max
Infravermelho próximo (0.851-0.879 µm)	B5_mean	B5_median	B5_stdDev	B5_min	B5_max
Infravermelho de ondas curtas (1.566-1.651 µm)	B6_mean	B6_median	B6_stdDev	B6_min	B6_max
Infravermelho de ondas curtas 2 (2.107-2.294 µm)	B7_mean	B7_median	B7_stdDev	B7_min	B7_max

Fonte: Do autor (2021).

Figura 9 - Imagem Landsat 8 usada na classificação em composição falsa-cor.



Fonte: Do autor (2021).

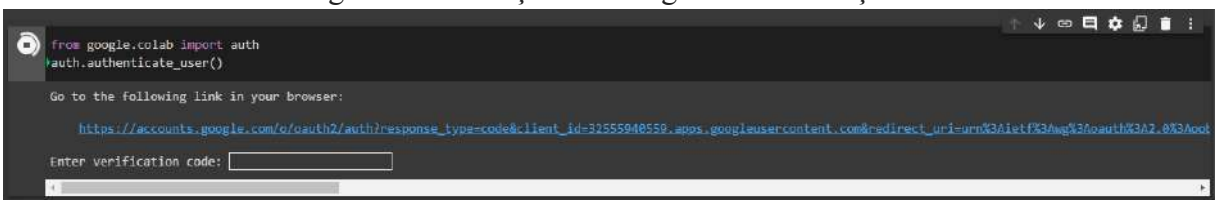
3.3.1 Execução do código

Para a realização do presente trabalho, foi utilizado um código disponibilizado pela empresa *Google LLC* (2020), que foi modificado a fim de atender os objetivos específicos da pesquisa. Esta seção irá expôr passo a passo cada célula de execução do código bem como suas respectivas saídas. O código foi implementado e executado em um *notebook* na plataforma *Google Colaboratory*.

3.3.1.1 Autenticação e instalação de bibliotecas

A utilização dos serviços *Google* faz-se possível após a autenticação em suas plataformas. O usuário deve possuir uma conta *Google* com acesso ao GEE e ao *Google Cloud Storage*, plataforma responsável por armazenar os dados usados no processamento em nuvem. A execução do código gerará um link que por sua vez irá gerar um código de autenticação. O código deve ser inserido na célula de execução para completar o processo de autenticação, como demonstrado nas Figuras 10 e 11.

Figura 10 - Geração do código de autenticação.



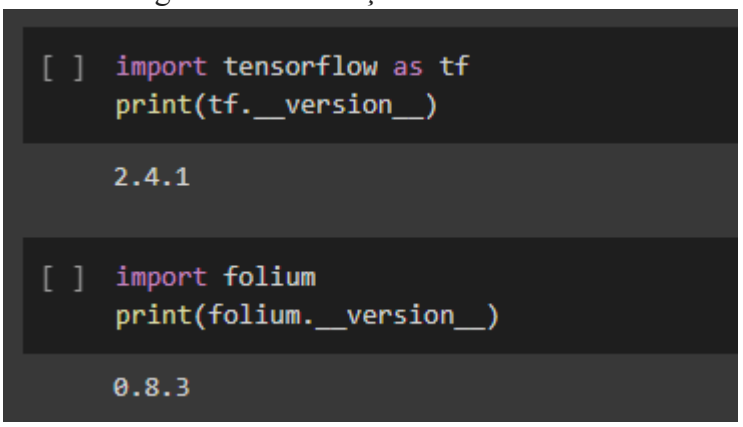
Fonte: Do autor (2021).

Figura 11- Autenticação completa.

Fonte: Do autor(2021).

O próximo passo é importar as bibliotecas que serão usadas no trabalho: *TensorFlow* para a criação de algoritmos de aprendizado de máquina e *Folium* para a visualização de resultados como demonstra a Figura 12.

Figura 12 - Instalação das bibliotecas.



Fonte: Do autor (2021).

3.3.1.2 Definição de variáveis

Esta etapa é responsável por definir as variáveis que serão usadas durante a execução do código, como: nome de usuário da plataforma *Google*, coleção de imagens que será usada, variáveis que serão usadas na predição, localização da base de dados de treinamento e teste do modelo, nome do rótulo de classes de cada ponto, número de classes, especificação das propriedades usadas pelo modelo, nome e localização dos arquivos de teste e treinamento, imagem final que será usada para predição, caminho de saída da imagem classificada, região usada na classificação e caminho e saída e nome do ativo classificado. As Figuras 13 e 14 demonstram o código usado.

Figura 13 - Definição das variáveis do algoritmo.

```
# Nome de usuário do Google Earth Engine. Usado para importar a imagem
# classificada para a pasta de ativos da plataforma.
USER_NAME = 'danielgacueto'

# Compartimento do Google Cloud Storage em que os dados de treinamento,
# teste e predições erão escritos.
OUTPUT_BUCKET = 'tensorflowtcc'

# Coleção de imagens que será usada no modelo
L8SR = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
# Bandas que serão usadas na predição
BANDS = ['B2_max', 'B3_max', 'B4_max', 'B5_max', 'B6_max', 'B7_max',
         'B2_median', 'B3_median', 'B4_median', 'B5_median', 'B6_median', 'B7_median',
         'B2_stdDev', 'B3_stdDev', 'B4_stdDev', 'B5_stdDev', 'B6_stdDev', 'B7_stdDev',
         'B2_min', 'B3_min', 'B4_min', 'B5_min', 'B6_min', 'B7_min',
         'B2_mean', 'B3_mean', 'B4_mean', 'B5_mean', 'B6_mean', 'B7_mean']

# Localização da base de dados dos pontos cuja classe de uso de solo já está
# definida. Usada para o teste e treinamento do algoritmo.
LABEL_DATA = ee.FeatureCollection('users/danielgacueto/PCTLand3C')
# Os rótulos de cada classe estão contidos nesta colçeção,
# georreferenciados em cada ponto coletado
LABEL = 'landcover'
# Número de rótulos, ou seja, números de classe na classificação.
N_CLASSES = 3
```

Fonte: Do autor (2021).

Figura 14 - Definição das variáveis do algoritmo.

```
# Esses nomes são usados para especificar propriedades na exportação de
# dados de treinamento / teste e para definir o mapeamento entre nomes e dados
# ao ler conjuntos de dados no formato TensorFlow.
FEATURE_NAMES = list(BANDS)
FEATURE_NAMES.append(LABEL)

# Nome dos arquivos para treinamento e teste.  Estes arquivos em formato
# TFRecord serão exportados do Earth Engine para o Cloud Storage.
TRAIN_FILE_PREFIX = 'TreinamentoLandsat3C'
TEST_FILE_PREFIX = 'TesteLandsat3C'
file_extension = '.tfrecord.gz'
TRAIN_FILE_PATH = 'gs://' + OUTPUT_BUCKET + '/' + TRAIN_FILE_PREFIX + file_extension
TEST_FILE_PATH = 'gs://' + OUTPUT_BUCKET + '/' + TEST_FILE_PREFIX + file_extension

# Nome da imagem final usada na predição.  O modelo treinado irá usar
# esta imagem como base e fazer predições em cada pixel da mesma.
IMAGE_FILE_PREFIX = 'Imagem_Landsat3C'

# Caminho de saída da imagem classificada em formato TFRecord.
OUTPUT_IMAGE_FILE = 'gs://' + OUTPUT_BUCKET + '/Imagem_Classificada_Landsat3C.TFRecord'
# Região para exportação da imagem.
EXPORT_REGION = ee.Geometry.Rectangle([-45.69318650159193, -19.17794777414055, -44.996927956670056, -18.372471950752168])

# Nome do arquivo a ser criado com a importação da imagem classificada
# no formato TFRecord para o Cloud Storage
OUTPUT_ASSET_ID = 'users/' + USER_NAME + '/Imagem_Classificada_Landsat3C'
```

Fonte: Do autor (2021).

3.3.1.3 Coleta dos dados para treinamento e teste do modelo

A coleta de dados para o treinamento do modelo foi feita manualmente. Os pontos foram coletados usando a interface GEE e rotulados com a propriedade "*landcover"* com valores variando de 0 a 5 para as diferentes classes utilizadas no algoritmo. Os pontos foram salvos em uma coleção de feições do GEE que contém as coordenadas geográficas de cada ponto e seu respectivo rótulo de classe. O Apêndice B contém o código fonte utilizado no processo.

3.3.1.4 Preparo da imagem e exportação das bases de dados para o formato TensorFlow

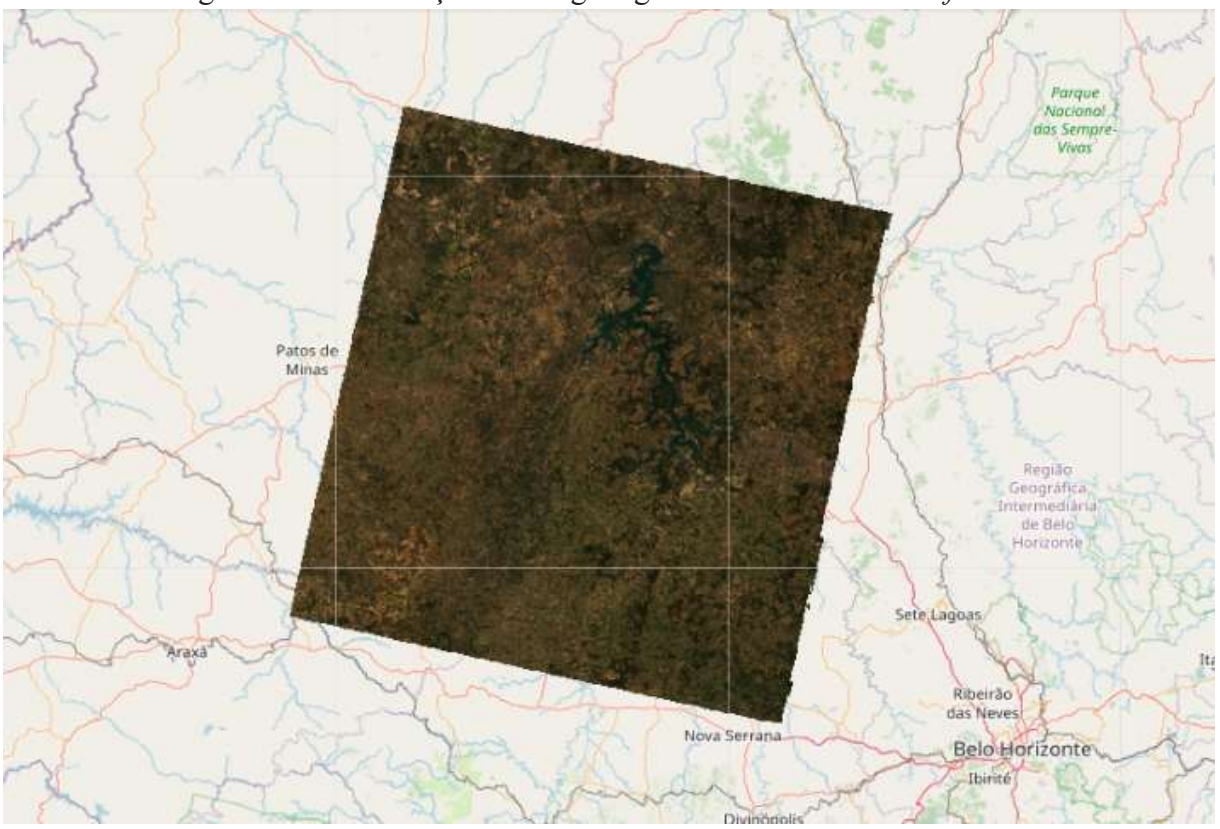
A imagem usada pelo modelo para realizar as predições foi criada a partir da coleção de imagens do satélite Landsat 8 adquiridas no ano de 2020. Inicialmente foi aplicada uma função para mascarar as nuvens da imagem. Em seguida, foi determinado o intervalo de tempo que define as imagens que serão utilizadas, no caso do presente trabalho, todas as imagens adquiridas em 2020. A biblioteca *folium* foi usada para visualizar a imagem criada e certificar que o processo ocorreu como esperado. As Figuras 15 e 16 demonstram este processo:

Figura 15 - Preparo da imagem.

```
# Função de máscara de nuvem
def maskL8sr(image):
  cloudShadowBitMask = ee.Number(2).pow(3).int()
  cloudsBitMask = ee.Number(2).pow(5).int()
  qa = image.select('pixel_qa')
  mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0).And(
    qa.bitwiseAnd(cloudsBitMask).eq(0))
  return image.updateMask(mask).select('B1','B2','B3','B4','B5','B6','B7').divide(10000)

# A imagem de entrada é um composto das estatísticas de bandas
# das imagens do ano de 2020 com a máscara de nuvens já aplicada
coll1 = L8SR.filterDate('2020-01-01', '2021-01-01').map(maskL8sr).filterBounds(EXPORT_REGION)
image = coll1.reduce(ee.Reducer.minMax().combine(reducer2=ee.Reducer.stdDev(),sharedInputs=True)
.combine(reducer2=ee.Reducer.median(),sharedInputs=True).combine(reducer2=ee.Reducer.mean(),sharedInputs=True))

# Use a biblioteca folium para visualizar a imagem formada e certificar que o processo ocorreu corretamente.
mapid = image.getMapId({'bands': ['B4_median', 'B3_median', 'B2_median'], 'min': 0, 'max': 0.3})

folium.TileLayer(
    tiles=mapid['tile_fetcher'].url_format,
    attr='Map Data &copy; <a href="https://earthengine.google.com/">Google Earth Engine</a>',
    overlay=True,
    name='median composite',
  ).add_to(map)
map.add_child(folium.LayerControl())
map
```

Fonte: Do autor (2021).

Figura 16- Visualização da imagem gerada com a biblioteca *folium*.



Fonte: Do autor (2021).

Com o preparo da imagem finalizado, a coleção de pontos de treinamento foi adicionada à mesma com o intuito de unir os rótulos de classe coletados aos pixels correspondentes na imagem. Os pontos coletados são usados para treinar e testar o modelo de forma aleatória, onde uma amostra de 70% dos pontos será usada para o treinamento e os 30% restantes para o teste do modelo. A Figura 17 demonstra este processo.

Figura 17 - Associação dos pontos de treinamento à imagem, repartição das amostras e verificação da saída do procedimento.

```
# Associar os dados coletados à imagem
sample = image.sampleRegions(
  collection=LABEL_DATA, properties=[LABEL], scale=30, tileScale=2).randomColumn()

# Repartir as amostras na proporção 70-30.
training = sample.filter(ee.Filter.lt('random', 0.7))
testing = sample.filter(ee.Filter.gte('random', 0.7))

from pprint import pprint

#Imprimir os primeiros pontos para verificar se o procedimento funcionou.
pprint({'training': training.first().getInfo()})
pprint({'testing': testing.first().getInfo()})
```

```
{'training': {'geometry': None,
              'id': '00000000000000000001_0',
              'properties': {'B1_max': 0.09000000357627869,
                             'B1_mean': 0.03288999944925308,
                             'B1_median': 0.0284000001847744,
                             'B1_min': 0.01899999938905239,
                             'B1_stdDev': 0.019561260184354175,
                             'B2_max': 0.10840000212192535,
                             'B2_mean': 0.04191000014543533,
                             'B2_median': 0.037050001323223114,
                             'B2_min': 0.023600000888109207,
                             'B2_stdDev': 0.02310612256524231,
                             'B3_max': 0.17260000109672546,
                             'B3_mean': 0.07596000283956528,
                             'B3_median': 0.06589999794960022,
                             'B3_min': 0.042399998754262924,
                             'B3_stdDev': 0.03608684524864981,
                             'B4_max': 0.19509999454021454,
                             'B4_mean': 0.06253000348806381,
                             'B4_median': 0.04904999956488609,
```

Fonte: Do autor (2021).

Com a imagem já processada e as bases de dados de treinamento e teste já definidas, é necessário exportar todas as informações de maneira que a biblioteca *TensorFlow* tenha acesso e consiga ler estes dados. Neste processo as bases de dados criadas até aqui serão convertidas para o formato *TFRecord* e adicionadas a um *bucket* do *Google Cloud Storage*. As Figuras 18 e 19 explicitam este processo.

Figura 18 - Exportação dos dados de teste e treinamento para o formato *TFRecord*.

```
# O bucket deve ser controlado pelo usuário, que deve poder ler e escrever no mesmo.
print('Found Cloud Storage bucket.' if tf.io.gfile.exists('gs://' + OUTPUT_BUCKET)
    else 'Can not find output Cloud Storage bucket.')

[ ] # Criando as tarefas de exportação
training_task = ee.batch.Export.table.toCloudStorage(
  collection=training,
  description='Training Export',
  fileNamePrefix=TRAIN_FILE_PREFIX,
  bucket=OUTPUT_BUCKET,
  fileFormat='TFRecord',
  selectors=FEATURE_NAMES)

testing_task = ee.batch.Export.table.toCloudStorage(
  collection=testing,
  description='Testing Export',
  fileNamePrefix=TEST_FILE_PREFIX,
  bucket=OUTPUT_BUCKET,
  fileFormat='TFRecord',
  selectors=FEATURE_NAMES)

[ ] # Iniciando as tarefas
training_task.start()
testing_task.start()

[ ] # Mostra as tarefas em andamento
pprint(ee.batch.Task.list())
```

Fonte: Do autor (2021).

Figura 19 - Exportação da imagem para o formato *TFRecord*.

```
[ ] # Exportação da imagem
image_export_options = {
  'patchDimensions': [256, 256],
  'maxFileSize': 104857600,
  'compressed': True
}

# Configurando a tarefa de exportação
image_task = ee.batch.Export.image.toCloudStorage(
  image=image,
  description='Image Export',
  fileNamePrefix=IMAGE_FILE_PREFIX,
  bucket=OUTPUT_BUCKET,
  scale=30,
  fileFormat='TFRecord',
  region=EXPORT_REGION.toGeoJSON()['coordinates'],
  formatOptions=image_export_options,
)

[ ] # Inicia as tarefas
image_task.start()

[ ] # Mostra as tarefas em andamento
pprint(ee.batch.Task.list())
```

Fonte: Do autor (2021).

3.3.1.5 Preparação e pré-processamento dos dados para uso com a biblioteca *TensorFlow*

As bases de dados exportadas no formato *TFRecord* devem ser pré-processadas para que fiquem em um formato adequado para inserção no modelo. É necessário criar uma função de análise dos arquivos exportados, cujo objetivo é indicar para a biblioteca *TensorFlow* como os dados em formato *TFRecord* serão transformados em tensores para leitura. Os dados estão em formato de matriz de duas dimensões. A primeira parte da matriz será usada para entrada no modelo e o último elemento da matriz como o rótulo da classe. O processo pode ser observado na Figura 20.

Figura 20 - Preparação e pré-processamento dos dados exportados.

```
# Criar um dataset a partir dos dados TFRecord exportados.
train_dataset = tf.data.TFRecordDataset(TRAIN_FILE_PATH, compression_type='GZIP')
# Imprimir os primeiros registros para conferir se estão certos.
print(iter(train_dataset).next())
columns = [
  tf.io.FixedLenFeature(shape=[1], dtype=tf.float32) for k in FEATURE_NAMES
]

features_dict = dict(zip(FEATURE_NAMES, columns))

pprint(features_dict)
```

```
def parse_tfrecord(example_proto):
  parsed_features = tf.io.parse_single_example(example_proto, features_dict)
  labels = parsed_features.pop(LABEL)
  return parsed_features, tf.cast(labels, tf.int32)

# Mapear a função em todo o dataset.
parsed_dataset = train_dataset.map(parse_tfrecord, num_parallel_calls=5)

# Imprimir os primeiros registros analisados para conferência.
pprint(iter(parsed_dataset).next())
```

Fonte: Do autor (2021).

O processo de preparação e pré-processamento dos dados é a última oportunidade de se adicionar novas feições à base de dados para ajudar na criação e treinamento do modelo. No presente trabalho, o índice de vegetação por diferença normalizada (NDVI) foi adicionado à base de dados. Meneses e Almeida (2012) pontuam que inúmeros trabalhos fizeram relação do NDVI com diversos aspectos da vegetação, como a medida de índice de área foliar, determinação da porcentagem da cobertura do solo e estimativas da radiação

fotossinteticamente ativa, que foram usados em vários modelos para estudos de fotossíntese e sequestro de carbono. O processo é demonstrado na Figura 21.

Figura 21- Adição do índice de vegetação com diferença normalizada (NDVI) à base de dados.

```
def normalized_difference(a, b):

  nd = (a - b) / (a + b)
  nd_inf = (a - b) / (a + b + 0.000001)
  return tf.where(tf.math.is_finite(nd), nd, nd_inf)

def add_NDVI(features, label):

  features['NDVI'] = normalized_difference(features['B5'], features['B4'])
  return features, label
```

Fonte: Do autor (2021).

3.3.1.6 Configuração, treinamento e execução do modelo

Os modelos de classificação no *TensorFlow* devem ser criados e treinados para que possam fazer inferências sobre a probabilidade que cada pixel possui de estar em uma determinada classe. No presente trabalho foi criado um modelo sequencial de rede neural profunda usando a API Keras.

Keras é uma API de alto nível usada pelo *TensorFlow* que permite a criação e treinamento de modelos de aprendizado profundo de maneira fácil e intuitiva. A API fornece uma interface simples e otimizada para usos comuns que fornece *feedback* claro e prático para os erros do usuário.

O modelo usado é chamado de modelo sequencial Keras que permite inserir as camadas da rede neural em série, onde o output da primeira camada serve como input da segunda, e assim por diante. Ele contará com duas camadas ocultas de 64 nós que usarão a função de ativação de unidade linear retificada (RelU), que, segundo os autores Ramachandran, Loph e Le (2017) é a função de ativação com maior sucesso e mais amplamente usada atualmente. O modelo também foi composto por uma camada de saída e uma camada *dropout*. A camada *dropout* escolhe aleatoriamente 20% dos dados de entrada do modelo em cada ciclo para serem descartados, com a finalidade de evitar o sobreajuste do mesmo. Foram realizados 80 ciclos de treinamento. Os números de ciclos de treinamento e número de nós na camada oculta foram escolhidos por tentativa e erro, tomando como partida

os métodos apresentados por Panchal *et al.* (2011), que evidencia que no final das contas a arquitetura da rede deve ser definida por tentativa e erro. A Figura 22 demonstra o processo:

Figura 22 - Definição das camadas, criação e compilação do modelo.

```
from tensorflow import keras

# Adicionar o NDVI criado à base de dados.
input_dataset = parsed_dataset.map(add_NDVI)

# Keras requer a entrada de dados em formato énuplo.
# Para a função de perda de entropia cruzada categorizada
# é necessário que os rótulos sejam transormados em um vetor one-hot.
def to_tuple(inputs, label):
  return (tf.transpose(list(inputs.values())),
          tf.one_hot(indices=label, depth=N_CLASSES))

input_dataset = input_dataset.map(to_tuple).batch(8)


# Definir as variáveis no modelo
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(64, activation=tf.nn.relu),
  tf.keras.layers.Dense(64, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(N_CLASSES, activation=tf.nn.softmax)
])

# Compilação do modelo com a função de perda escolhida.
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Ajuste do modelo à base de dados.
model.fit(x=input_dataset, epochs=80)
```

Fonte: Do autor (2021).

Com o modelo treinado, sua acurácia foi testada usando o conjunto de dados de teste. Esta acurácia calculada não é uma representação fiel da acurácia calculada da imagem já classificada. Ela é usada para analisar como o modelo se comporta na hora de ajustar o número de camadas e nós usados no algoritmo pelo processo de tentativa e erro. É necessário preparar o conjunto de dados de teste da mesma maneira que os dados de treinamento, com a diferença de definir o tamanho de lote para 1 com o intuito que cada dado do conjunto de teste seja usado uma única vez no cálculo da precisão do modelo, como demonstrado na Figura 23.

Figura 23 - Checagem de acurácia do modelo usando a base de dados de teste.

```
test_dataset = (
  tf.data.TFRecordDataset(TEST_FILE_PATH, compression_type='GZIP')
    .map(parse_tfrecord, num_parallel_calls=5)
    .map(add_NDVI)
    .map(to_tuple)
    .batch(1))

model.evaluate(test_dataset)
```

Fonte: Do autor (2021).

Concluídos o treinamento e teste do modelo, a imagem exportada do GEE foi classificada. Nesta etapa, faz-se necessário especificar o diretório cujos arquivos foram exportados. Junto com os arquivos da base de dados em formato *TFRecord* foram criados arquivos auxiliares contendo a metadata dos respectivos arquivos, como o formato original da imagem e seu georreferenciamento. O processo é demonstrado na Figura 24.

Figura 24- Localização dos arquivos base e auxiliares.

```
# Lista todos os arquivos presentes no bucket diretório.
files_list = !gsutil ls 'gs://'{OUTPUT_BUCKET}
# Seleciona apenas arquivos gerados na função de exportação das imagens
exported_files_list = [s for s in files_list if IMAGE_FILE_PREFIX in s]

# Lista todos arquivos com seus respectivos anexos JSON.
image_files_list = []
json_file = None
for f in exported_files_list:
  if f.endswith('.tfrecord.gz'):
    image_files_list.append(f)
  elif f.endswith('.json'):
    json_file = f

# Coloca os arquivos na ordem correta.
image_files_list.sort()

pprint(image_files_list)
print(json_file)
```

```
import json
# Anexa a metadata dos arquivos JSON
json_text = !gsutil cat {json_file}
mixer = json.loads(json_text.nlstr)
pprint(mixer)
```

Fonte: Do autor (2021).

As imagens exportadas devem ser adicionadas em um *dataset TensorFlow* para que o modelo possa fazer inferências sobre a classificação. A entrada deste *dataset* precisa ser

pré-processada de forma diferente do treinamento e teste. Isso se dá pelo fato que os pixels são registrados em sequência nesta etapa e estas sequências devem ser lidas em um único longo tensor totalizando uma sequência por banda da imagem. Este tensor longo com todo o *dataset* foi quebrado em vários pequenos tensores posteriormente para a realização da predição dos resultados. As Figuras 25 e 26 representam o processo:

Figura 25 - Inserção da imagem no formato do *dataset TensorFlow*.

```
# Extrai informações do arquivo JSON.
patch_width = mixer['patchDimensions'][0]
patch_height = mixer['patchDimensions'][1]
patches = mixer['totalPatches']
patch_dimensions_flat = [patch_width * patch_height, 1]

# Os tensores estão em sequência, uma sequência por banda da imagem.
image_columns = [
  tf.io.FixedLenFeature(shape=patch_dimensions_flat, dtype=tf.float32)
    for k in BANDS
]

# Chaves da função de análise
image_features_dict = dict(zip(BANDS, image_columns))

# É possível fazer um dataset a partir de várias imagens especificadas na lista
image_dataset = tf.data.TFRecordDataset(image_files_list, compression_type='GZIP')

# Função de análise
def parse_image(example_proto):
  return tf.io.parse_single_example(example_proto, image_features_dict)

# Transformar o dataset em um longo tensor por sequência de dados
image_dataset = image_dataset.map(parse_image, num_parallel_calls=5)

# Quebrar o tensor longo em vários menores.
image_dataset = image_dataset.flat_map(
  lambda features: tf.data.Dataset.from_tensor_slices(features)
)

# Adicionar o NDVI
image_dataset = image_dataset.map(
  # NDVI deve ser atrelado à uma função ainda sem rótulo.
  lambda features: add_NDVI(features, None)[0]
```

Fonte: Do autor (2021).

Figura 26 - Geração das predições em lote.

```
# Transforma as chaves da função de análise em uma sequência énupla.
image_dataset = image_dataset.map(
  lambda data_dict: (tf.transpose(list(data_dict.values())), )
)

# Transforma cada sequência em um lote de processamento
image_dataset = image_dataset.batch(patch_width * patch_height)

# Executa a predição em lotes, usando o número de passos igual ao número de lotes.
predictions = model.predict(image_dataset, steps=patches, verbose=1)

# As predições são geradas em forma de matriz.
print(predictions[0])
```

Fonte: Do autor (2021).

3.3.1.7 Transformação das predições geradas em imagem

A execução do modelo gerou uma lista da probabilidade que cada pixel possui de estar em uma determinada classe. A transformação foi feita adicionando o rótulo da classe com maior probabilidade à cada pixel da imagem. A imagem gerada possui um número de bandas igual ao número de classes usadas no modelo +1. Cada banda contém as informações da probabilidade que cada pixel da imagem possui de estar em uma dada classe e a banda adicional contém o rótulo de classe da maior probabilidade em cada pixel. O processo é demonstrado na Figura 27.

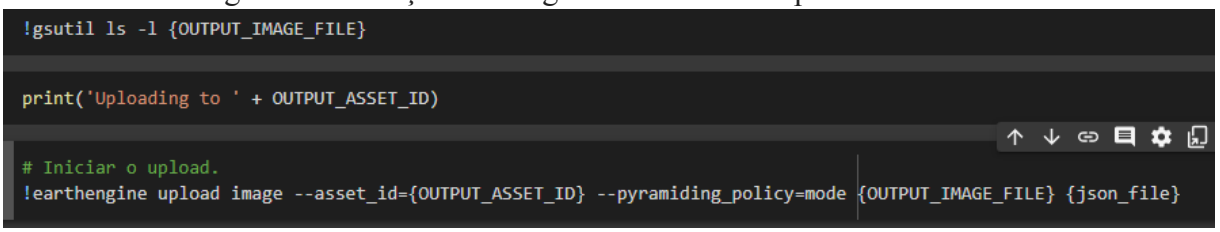
Figura 27 - Transformação das predições em imagem.

```
# Iniciar a transformação.
writer = tf.io.TFRecordWriter(OUTPUT_IMAGE_FILE)

# Cada sequência de predições irá adicionar uma feição com as predições geradas
# no arquivo de saída. Como as predições já foram ordenadas em sequência anteriormente,
# as sequências criadas aqui já estarão na ordem certa
patch = [[], [], [], []]
cur_patch = 1
for prediction in predictions:
  patch[0].append(tf.argmax(prediction, 1))
  patch[1].append(prediction[0][0])
  patch[2].append(prediction[0][1])
  patch[3].append(prediction[0][2])
  # Assim que uma sequência de classes finalizar...
  if (len(patch[0]) == patch_width * patch_height):
    print('Done with patch ' + str(cur_patch) + ' of ' + str(patches) + '...')
    # Cria-se um exemplo
    example = tf.train.Example(
      features=tf.train.Features(
        feature={
          'prediction': tf.train.Feature(
              int64_list=tf.train.Int64List(
                  value=patch[0])),
          'bareProb': tf.train.Feature(
              float_list=tf.train.FloatList(
                  value=patch[1])),
          'vegProb': tf.train.Feature(
              float_list=tf.train.FloatList(
                  value=patch[2])),
          'waterProb': tf.train.Feature(
              float_list=tf.train.FloatList(
                  value=patch[3])),
        }
      )
    )
    # Os exemplos são registrados no arquivo de saída as sequências são reorganizadas
    # para mais um lote ser escrito.
    writer.write(example.SerializeToString())
    patch = [[], [], [], []]
    cur_patch += 1

writer.close()
```

Fonte: Do autor (2021).

A imagem final gerada está em formato *TFRecord* e deve ser inserida na plataforma GEE usando o arquivo auxiliar JSON, que contém os dados de georreferenciamento da imagem, para que seja visualizada. Com este processo ela foi transformada em um ativo da plataforma permitindo sua manipulação, análise e *download* em formatos mais convencionais. O processo é demonstrado na Figura 28.

Figura 28- Inserção da imagem como ativo da plataforma GEE.



Fonte: Do autor (2021).

3.3.2 Análise dos resultados

A análise da acurácia da classificação de uso e cobertura do solo foi realizada por meio de uma matriz de confusão. Segundo Banko (1998), a matriz de confusão é a maneira mais comum de se expressar a acurácia de uma classificação. Ela consiste em uma tabulação cruzada simples das classes alocadas à classificação contra os dados de referência. A matriz de erros organiza os dados de amostra adquiridos com o intuito de resumir os resultados e auxiliar na análise da eficácia do método utilizado e do grau de erro da classificação final gerada (OLOFSSON *et al.*, 2014).

A diagonal principal da matriz de erro mostra as classificações corretas enquanto os elementos fora da diagonal mostram erros de omissão e comissão em cada classe. A análise da diagonal principal traz o índice de acurácia geral da classificação que é calculado realizando a divisão dos pixels classificados corretamente pelo número total de pixels analisados. Banko (1998) pontua que os erros de omissão e comissão estão atrelados a duas outras medidas de acurácia: a acurácia do produtor e a acurácia do consumidor respectivamente. A acurácia do produtor mede o quão bem uma determinada área foi classificada. Inclui o erro de omissão que se refere à proporção de feições observadas no terreno que não foram classificadas na imagem. A acurácia do consumidor diz respeito à confiabilidade da imagem classificada. Ela informa quão bem a imagem representa o que está em campo.

Desenvolvido por Cohen (1960) a última medida de acurácia calculada será o coeficiente Kappa que possui a finalidade de medir o grau de concordância (confiabilidade e precisão) da classificação. Landis e Koch (1977) definiram faixas de valores do índice que se associam com a qualidade da classificação, como demonstra a Tabela 4.

Tabela 4 - Qualidade da classificação associada ao valor do coeficiente Kappa.

Coeficiente Kappa	Qualidade da classificação
0,00	Péssima
0,01 a 0,20	Ruim
0,21 a 0,40	Razoável
0,41 a 0,60	Boa
0,61 a 0,80	Muito Boa
0,81 a 1,00	Excelente

Fonte: Adaptado de Landis e Koch (1977).

Com o intuito de realizar uma análise minuciosa da acurácia da rede, foram observados os índices de acurácia de 8 classificações geradas utilizando o algoritmo apresentado. Primeiramente, foram geradas 4 classificações da área de estudo com números distintos de classes, variando de 3 a 6, para averiguar o comportamento da Rede Neural criada em cada uma das situações. Posteriormente, outras 4 classificações foram geradas usando os mesmos números de classes e as mesmas amostras de treinamento e validação das primeiras, porém com uma base de dados de entrada reduzida para se avaliar o efeito da quantidade e complexidade da base de dados de entrada na acurácia final do algoritmo. Os modelos comparativos tiveram como base uma imagem contendo apenas a mediana das bandas do visível, infravermelho próximo e infravermelho de ondas curtas do satélite Landsat 8 no ano de 2020 como dados de entrada, ou seja, foi usada uma imagem com o total de 6 bandas, diferente das primeiras classificações que contaram com 30 bandas cada. As classes usadas em cada modelo são demonstradas na Tabela 5.

Tabela 5 - Classes utilizadas em cada classificação gerada.

Modelo	Classes utilizadas
Modelo 1	3 classes - Água, vegetação e solo exposto+áreas urbanas
Modelo 2	4 classes - Água, vegetação nativa, agricultura+pastagens e solo exposto+áreas urbanas.
Modelo 3	5 classes - Água, vegetação nativa, agricultura+pastagens, solo exposto, áreas urbanas
Modelo 4	6 classes - Água, vegetação nativa, agricultura, pastagens, solo exposto e áreas urbanas

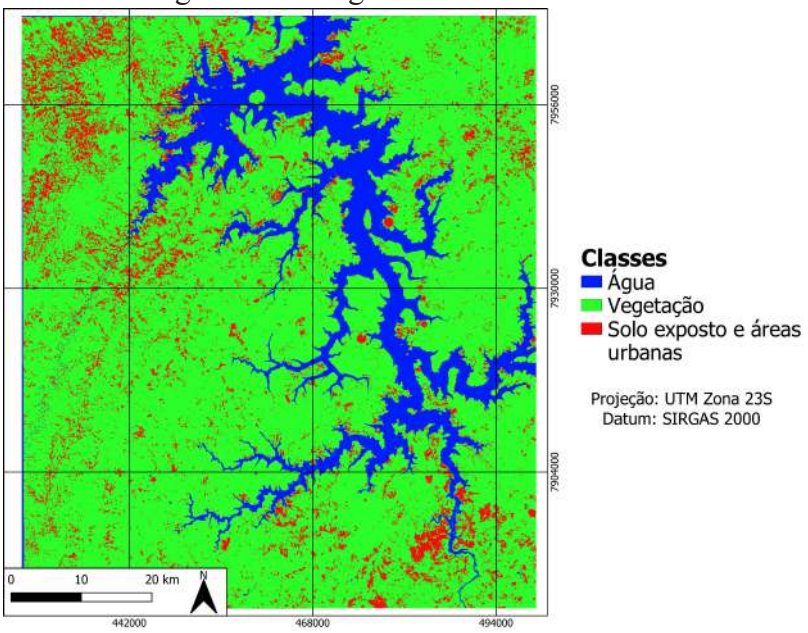
Fonte: Do autor (2021).

4 RESULTADOS E DISCUSSÃO

A partir da execução dos códigos demonstrados na plataforma *Google Colab* foram geradas as classificações de uso e cobertura do solo da área de estudo.

O modelo 1 contou com 112 pontos de treinamento da classe água, 377 de vegetação e 265 para solo exposto e áreas urbanas, todos bem distribuídos na extensão da imagem. A amostra usada como verdade de campo na matriz de confusão contou com 793 pontos distribuídos de maneira uniforme na imagem. A Figura 29 mostra a classificação gerada.

Figura 29 - Imagem classificada do modelo 1.



Fonte: Do autor (2021).

As classes analisadas neste modelo são de simples distinção e o modelo apresentou alta acurácia e precisão, como demonstra a Tabela 6.

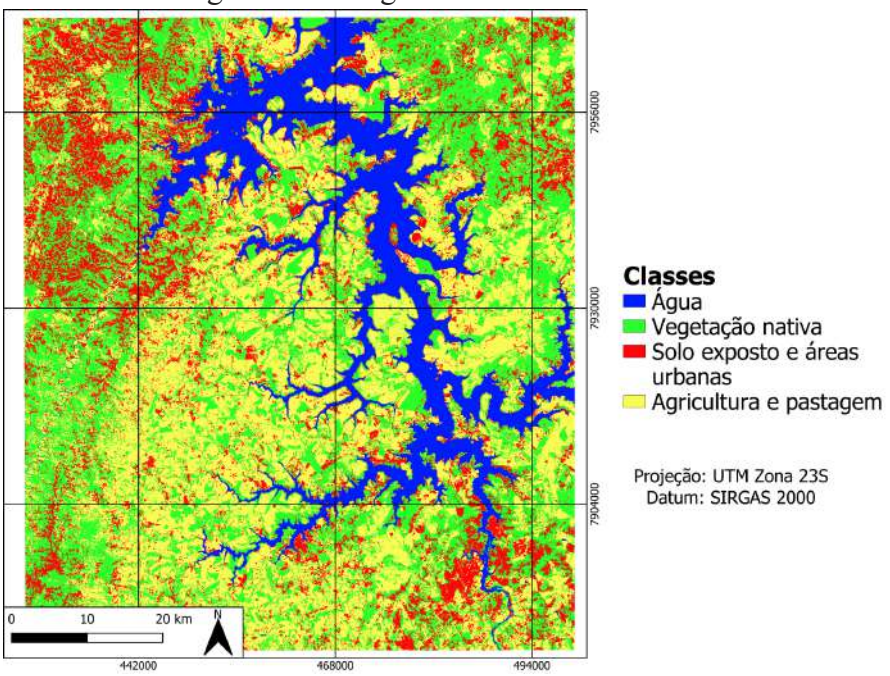
Tabela 6 - Matriz de confusão, acurácia global e coeficiente Kappa do modelo 1.

		Classe prevista			
		Água	Vegetação	Solo exposto e áreas urbanas	Acurácia do produtor
Classe real	Água	186	0	0	100%
	Vegetação	0	421	30	93,3%
	Solo+urbano	0	24	132	84,6%
	Acurácia do consumidor	100%	94,6%	81,5%	Global: 93,1%
Coeficiente Kappa: 0,88					

Fonte: Do autor (2021).

O treinamento do modelo 2 foi realizado com a coleta de 136 pixels da classe água, 259 da classe vegetação nativa, 368 da classe solo exposto e áreas urbanas e 282 pontos da classe agricultura e pastagens. A amostra usada como verdade de campo na matriz de confusão contou com 897 pontos distribuídos de maneira uniforme na imagem. A Figura 30 mostra a classificação gerada e a Tabela 7 expõe os indicadores de acurácia do modelo.

Figura 30 - Imagem classificada do modelo 2.



Fonte: Do autor (2021)

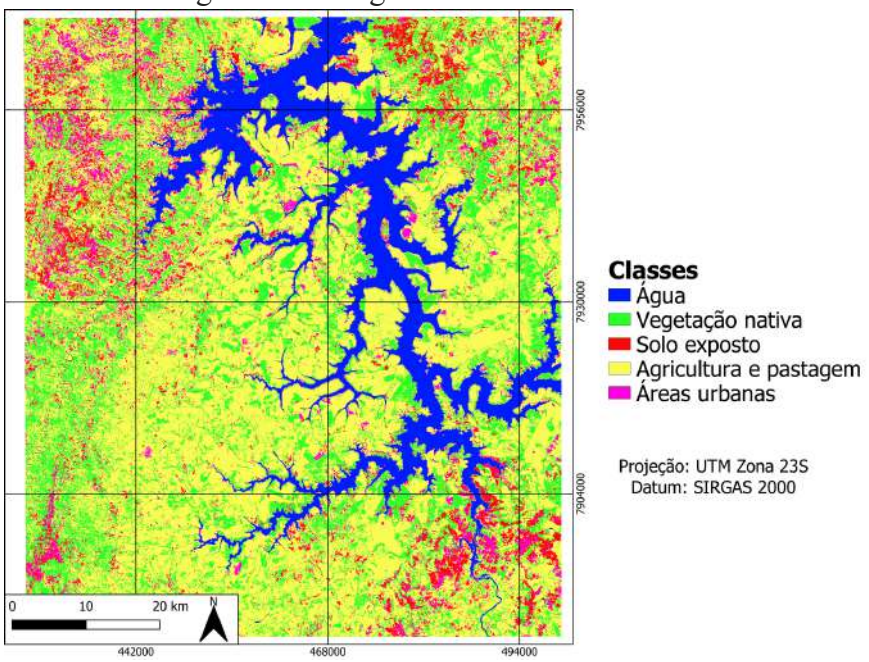
Tabela 7 - Matriz de confusão, acurácia global e coeficiente Kappa do modelo 2.

		Classe prevista				
		Água	Vegetação nativa	Solo+Urbano	Agri+Pasto	Acurácia do produtor
Classe real	Água	179	0	0	0	100%
	Vegetação nativa	1	203	8	33	82,8%
	Solo+urbano	0	8	169	12	89,4%
	Agri+Pasto	0	27	31	226	79,5%
	Acurácia do consumidor	99,4%	85,2%	81,2%	83,3%	Global: 86,6%
Coeficiente Kappa: 0,82						

Fonte: Do autor (2021).

O treinamento do modelo 3 foi realizado com a coleta de 112 pixels da classe água, 226 da classe vegetação nativa, 260 da classe solo exposto, 396 pontos da classe agricultura e pastagens e 106 pontos em áreas urbanas. A amostra usada como verdade de campo na matriz de confusão contou com 918 pontos distribuídos de maneira uniforme na imagem. A Figura 31 mostra a classificação gerada e a Tabela 8 expõe os indicadores de acurácia do modelo.

Figura 31 - Imagem classificada do modelo 3.



Fonte: Do autor (2021).

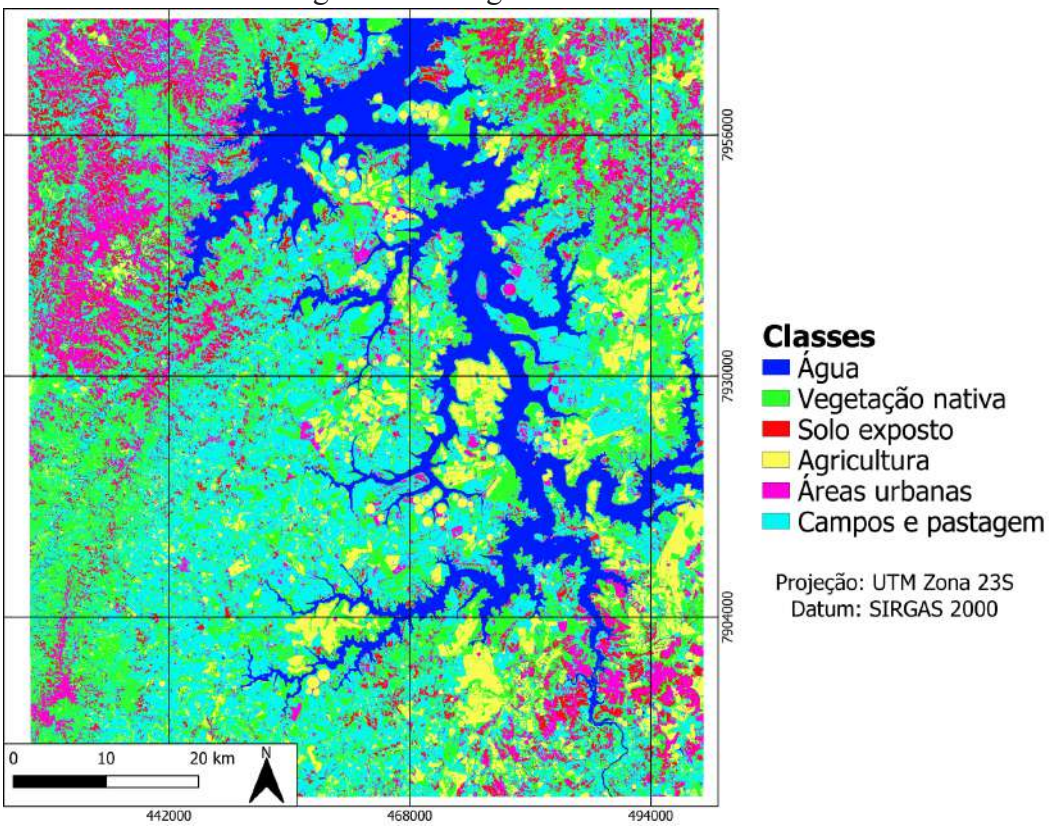
Tabela 8 - Matriz de confusão, acurácia global e coeficiente Kappa do modelo 3.

		Classe prevista					
		Água	Vegetação nativa	Solo Exposto	Agri+Pasto	Urbano	Acurácia do produtor
Classe real	Água	173	0	0	0	0	100%
	Vegetação nativa	0	62	0	65	2	48,0%
	Solo exposto	0	1	149	44	50	60,9%
	Agri+Pasto	0	11	0	204	25	85,0%
	Urbano	0	1	6	8	117	88,4%
	Acurácia do consumidor	100%	82,0%	95,4%	63,5%	59,2%	Global: 76,4%
Coeficiente Kappa: 0,71							

Fonte: Do autor (2021).

O treinamento do modelo 4 foi realizado com a coleta de 112 pixels da classe água, 243 da classe vegetação nativa, 289 da classe solo exposto, 187 pontos da classe agricultura, 178 pontos em áreas urbanas e 266 pontos em áreas de campo ou pastagem. A amostra usada como verdade de campo na matriz de confusão contou com 1281 pontos distribuídos de maneira uniforme na imagem. A Figura 32 mostra a classificação gerada e a Tabela 9 expõe os indicadores de acurácia do modelo.

Figura 32 - Imagem classificada do modelo 4.



Fonte: Do autor (2021).

Tabela 9 - Matriz de confusão, acurácia global e coeficiente Kappa do modelo 4.

		Classe prevista						
		Água	Vegetação Nativa	Solo Exposto	Agricultura	Urbano	Pasto	Acurácia do produtor
Classe real	Água	176	0	0	0	0	0	100%
	Vegetação Nativa	0	101	0	32	4	17	65,5%
	Solo exposto	0	3	217	1	123	27	58,9%
	Agricultura	0	31	4	118	9	51	55,3%
	Urbano	0	0	8	0	126	6	90,0%
	Pasto	0	6	7	19	5	193	83,9%
	Acurácia do consumidor	100%	73,1%	91,9%	69,4%	47,1%	65,6%	Global: 72,6%
Coeficiente Kappa: 0,67								

Fonte: Do autor (2021).

A análise dos resultados mostra que a Rede Neural criada apresentou um coeficiente Kappa calculado considerado excelente pela literatura (>0,81) para a classificação de 3 e 4 classes de uso e cobertura do solo, porém sua acurácia decresceu à medida que mais classes foram utilizadas no modelo, obtendo-se um coeficiente Kappa considerado muito bom (0,61-0,80) para a distinção de 5 e 6 diferentes classes de cobertura do solo.

A acurácia do produtor e do consumidor demonstram o comportamento da qualidade da classificação em cada classe. A classe água foi a com melhor acurácia de classificação devido à sua assinatura espectral singular. A rede encontrou maior dificuldade em distinguir *pixels* de solo exposto dos de áreas urbanas, provavelmente devido ao fato que as cidades presentes na cena são pouco urbanizadas e muito arborizadas, fazendo com que sua assinatura espectral se assemelhe à do solo exposto e de formações rochosas com vegetação emergente. A classe de vegetação nativa também foi muitas vezes classificada erroneamente pela rede como agricultura, devido ao fato de ambas apresentarem assinatura espectral semelhante.

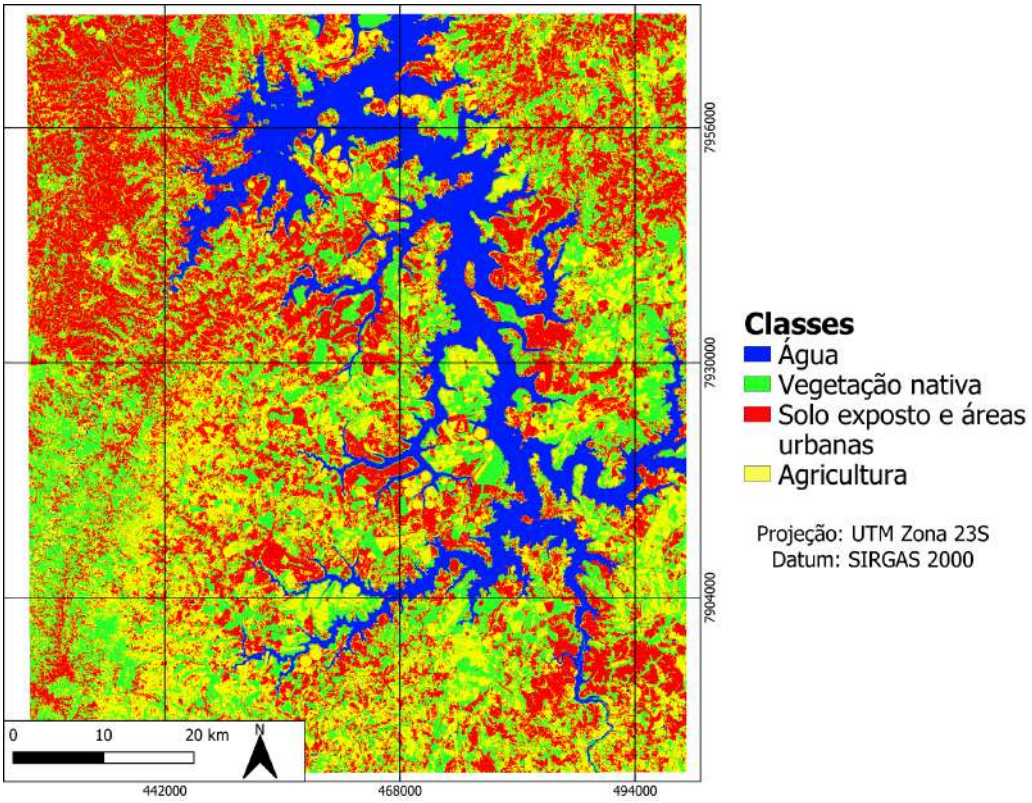
Srivastava (2012) demonstra em seu trabalho a comparação de diferentes técnicas de classificação do uso e cobertura do solo para a distinção de 5 classes em uma área no estado de Iowa - Estados Unidos da América, uma delas sendo a implementação de uma Rede Neural Artificial. Em seu estudo é demonstrado que a RNA obteve um coeficiente Kappa calculado de 0,74 para a classificação de 5 classes em sua área de estudo, índice similar ao obtido no presente trabalho para o mesmo número de classes, que foi de 0,71. Comparativamente, o estudo de Hu et al. (2018) determina um coeficiente Kappa de 0,76 para a classificação de 7 distintas classes e Silva et al. (2020) obtém um coeficiente Kappa de 0,77 para a distinção de 8 classes usando uma RNA.

Estes resultados, apesar de não serem diretamente comparáveis, evidenciam a complexidade das interações entre a arquitetura, a base de dados de entrada e treinamento e a qualidade final dos produtos gerados por este tipo de algoritmo. Apesar de uma das características das Redes Neurais Artificiais ser a generalização, a relação dos atributos usados com a estrutura, arquitetura e qualidade geral da rede possui grande especificidade para cada distinta aplicação.

Por fim, com o intuito de avaliar o efeito da complexidade e quantidade dos dados de entrada no modelo, quatro últimas classificações comparativas foram geradas usando uma base de dados de entrada reduzida. Nadikattu (2017) evidencia que quanto maior a quantidade de dados que podem ser enviados para uma rede neural, mais precisa ela será. Todos os modelos comparativos obtiveram índices de acurácia global e coeficiente Kappa inferior às suas contrapartes. Para efeito de demonstração, os modelos comparativos 2 e 3, que

apresentaram maiores discrepâncias visuais em suas classificações, são expostos nas Figuras 33 e 34 e suas matrizes de confusão nas Tabelas 10 e 11 respectivamente.

Figura 33 - Imagem classificada pelo modelo comparativo ao modelo 2.



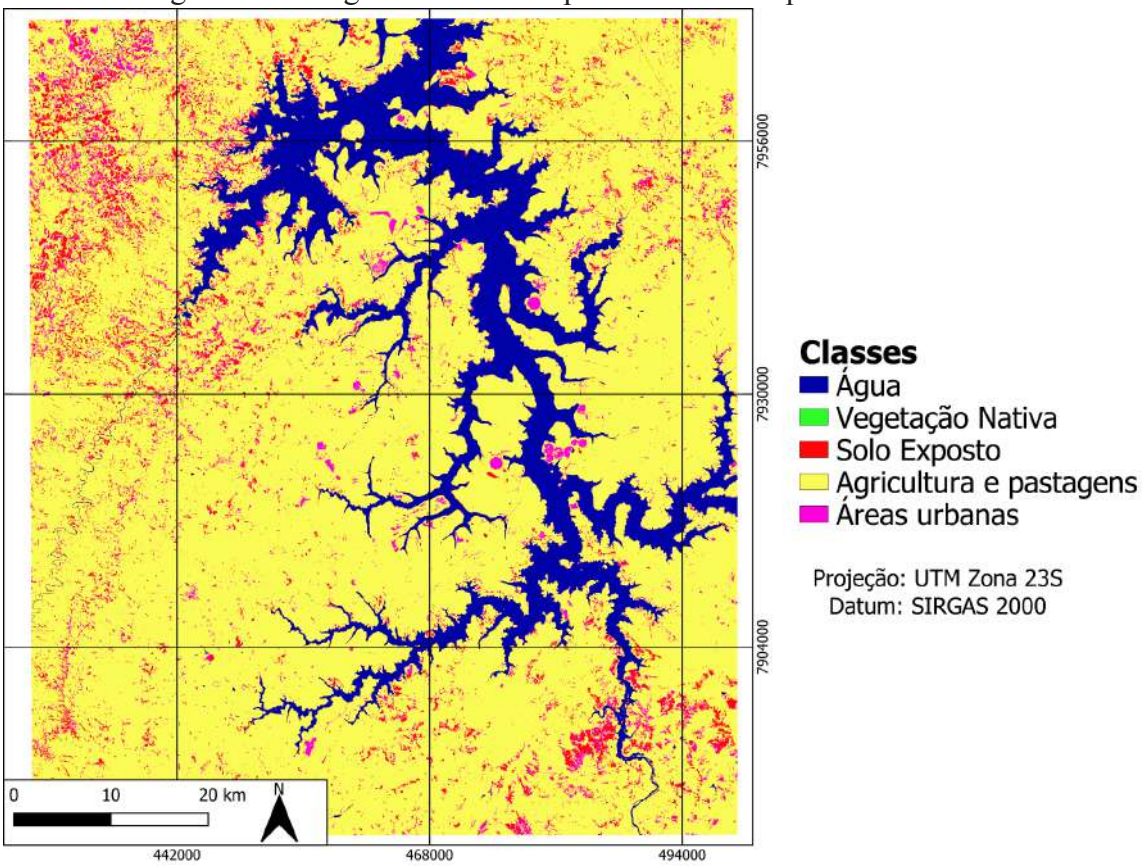
Fonte: Do autor (2021).

Tabela 10 - Matriz de confusão, acurácia global e coeficiente Kappa da classificação comparativa ao modelo 2.

		Classe prevista				
		Água	Vegetação nativa	Solo+Urbano	Agri+Pasto	Acurácia do produtor
Classe real	Água	179	0	0	0	100%
	Vegetação nativa	0	195	10	40	79,5%
	Solo+urbano	0	0	183	6	96,8%
	Agri+Pasto	0	36	96	152	53,5%
	Acurácia do consumidor	100%	84,4%	63,3%	76,7%	Global: 79,0%
Coeficiente Kappa: 0,72						

Fonte: Do autor (2021).

Figura 34 - Imagem classificada pelo modelo comparativo ao modelo 3.



Fonte: Do autor (2021).

Tabela 11 - Matriz de confusão, acurácia global e coeficiente Kappa da classificação comparativa ao modelo 3.

		Classe prevista					
		Água	Vegetação nativa	Solo Exposto	Agri+Pasto	Urbano	Acurácia do produtor
Classe real	Água	176	0	0	0	0	100%
	Vegetação nativa	0	0	0	127	2	0%
	Solo exposto	0	0	152	43	53	61,2%
	Agri+Pasto	0	0	0	217	13	94,3%
	Urbano	0	0	9	16	105	80,7%
	Acurácia do consumidor	100%	0%	94,4%	53,8%	60,6%	Global: 71,1%%
Coeficiente Kappa: 0,62							

Fonte: Do autor (2021).

Apenas com a análise visual das últimas classificações é perceptível que a classe de solo exposto e áreas urbanas foi superestimada em comparação ao modelo 2 e o mesmo aconteceu com a classe de agricultura em comparação ao modelo 3. Devido à baixa complexidade dos dados de entrada, ocorreu o sobreajuste do modelo e nenhum pixel da imagem foi considerado pertencente à classe de vegetação nativa pelo algoritmo no modelo comparativo 3. A Tabela 12 demonstra a comparação dos índices de acurácia dos modelos originais e comparativos.

Tabela 12 - Comparação dos modelos originais e comparativos.

Modelo	**Modelo original**		**Modelo comparativo**	
	Acurácia Global	**Coeficiente Kappa**	**Acurácia Global**	**Coeficiente Kappa**
1	93,1%	0,88	88,4% (-5,0%)	0,80 (-9,1%)
2	86,6%	0,82	79,0% (-8,7%)	0,72 (-12,1%)
3	76,4%	0,71	71,1% (-6,9%)	0,62 (-12,6%)
4	72,6%	0,67	65,2% (-10,1%)	0,57 (-14,9%)

Fonte: Do autor (2021).

Observa-se que tanto a acurácia global quanto o coeficiente Kappa apresentaram uma queda considerável em todos os modelos e foi maior nos modelos com mais classes, o que mostra que a quantidade e complexidade dos dados de entrada foram fatores determinantes na qualidade das classificações geradas pela Rede Neural Artificial criada, principalmente quando usada para a distinção de mais classes.

5 CONCLUSÃO

O estudo buscou demonstrar detalhadamente o processo realizado na implementação do modelo Keras sequencial da biblioteca *TensorFlow* em conjunto com a plataforma *Google Colab* e *Google Earth Engine* para criação de uma Rede Neural Artificial Profunda. Por meio dos algoritmos desenvolvidos foi possível implementar uma Rede Neural Artificial usada para classificação de imagens de satélite que apresentou uma boa acurácia na distinção de 3 e 4 classes porém teve seu desempenho prejudicado à medida que mais classes foram adicionadas. Os modelos gerados na pesquisa apresentaram coeficiente Kappa considerados muito bons pela literatura (>0,61), porém ainda observou-se muita sub e superestimação de algumas fisionomias com a análise visual das classificações.

A importância da quantidade, qualidade e complexidade dos dados de entrada na Rede Neural Artificial desenvolvida foi demonstrada por meio dos modelos comparativos desenvolvidos, onde todos os modelos executados com a base de dados de entrada reduzida apresentaram índices de acurácia inferiores à sua contraparte. Ademais, buscou-se expor detalhadamente os processos e ferramentas disponíveis gratuitamente para qualquer pessoa que tenha necessidade ou desejo de trabalhar com grandes bases de dados de sensoriamento remoto e aprendizado de máquina.

A adição de novas bases de dados no modelo como imagens de melhor resolução, bases topográficas, climáticas, sensores termais ou qualquer informação que seja pertinente à aplicação desenvolvida é possível e deve ser alvo de futuras pesquisas. Por fim, torna-se necessário a continuidade dos estudos relacionando a acurácia da rede com sua arquitetura para uma definição mais refinada de seus parâmetros de configuração, de como seus nós se interconectam, seus pesos e de como cada camada responde ao *output* da outra para obtenção de melhores resultados.

REFERÊNCIAS

ABADI, M. *et al*. Tensorflow: A system for large-scale machine learning. In: **12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)**. p. 265-283, 2016a.

ABADI, M. *et al*. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **arXiv preprint arXiv:1603.04467**, 2016b.

BANKO, G. **A review of assessing the accuracy of classifications of remotely sensed data and of methods including remote sensing data in forest inventory.** 1998.

BATISTA, G. E. A. P. *et al*. **Pré-processamento de dados em aprendizado de máquina supervisionado**. Tese (Doutorado) - Universidade de São Paulo. 2003.

BIOUCAS-DIAS, J. M. et al. Hyperspectral remote sensing data analysis and future challenges. **IEEE Geoscience and remote sensing magazine**, v. 1, n. 2, p. 6-36, 2013.

BRECHIN, S. R.; BHANDARI, Medani. Perceptions of climate change worldwide. **Wiley Interdisciplinary Reviews: Climate Change**, v. 2, n. 6, p. 871-885, 2011.

CARVALHO, A. *et al*. **Inteligência Artificial: Uma abordagem de Aprendizado de Máquina.** Rio de Janeiro: Ltc, v. 2, p. 192, 2011.

CHI, M. *et al*. Big data for remote sensing: Challenges and opportunities. **Proceedings of the IEEE**, v. 104, n. 11, p. 2207-2219, 2016.

COHEN, J. A coefficient of agreement for nominal scales. **Educational and psychological measurement**, v. 20, n. 1, p. 37-46, 1960.

CYBENKO, G. **Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems**, v. 2, n. 4, p. 303-314, 1989.

EUCLYDES, H. P. *et al*. Regionalização hidrológica na bacia do alto São Francisco a montante da barragem de Três Marias, Minas Gerais. **Revista Brasileira de Recursos Hídricos**, v. 6, n. 2, p. 81-105, 2001.

FIGUEIREDO, D. Conceitos básicos de sensoriamento remoto. **São Paulo**, 2005.

GATES, D. M. *et al*. Spectral properties of plants. **Applied optics**, v. 4, n. 1, p. 11-20, 1965.

GOLDSCHMIDT, R. R. Uma Introdução à Inteligência Computacional: fundamentos, ferramentas e aplicações. **Rio de Janeiro Brasil: IST-Rio**, v. 1, p. 32, 2010.

GORELICK, N. *et al*. Google Earth Engine: Planetary-scale geospatial analysis for everyone. **Remote sensing of Environment**, v. 202, p. 18-27, 2017.

HU, Y. *et al*. **A deep convolutional neural network method for land cover mapping: A case study of Qinhuangdao, China.** Remote Sensing, v. 10, n. 12, p. 2053, 2018.

JENSEN, J. R.; EPIPHANIO, J. C. N. **Sensoriamento remoto do ambiente: uma perspectiva em recursos terrestres**. São José dos Campos: Parêntese Editora, 2009.

JENSEN, J. R. Biophysical remote sensing. **Annals of the Association of American Geographers**, v. 73, n. 1, p. 111-132, 1983.

LANDIS, J. R.; KOCH, G. G. The measurement of observer agreement for categorical data. **Biometrics**, p. 159-174, 1977.

LILLESAND, T.; KIEFER, R. W.; CHIPMAN, J. **Remote sensing and image interpretation**. John Wiley & Sons, 2015.

MACHADO, C. A. S.; QUINTANILHA, J. A. Módulo de treinamento: Sistemas de Informações Geográficas (SIG) e Geoposicionamento: Uma aplicação Urbana. **São Paulo: USP**, 2008.

MAHDIANPARI, M. *et al*. The first wetland inventory map of Newfoundland at a spatial resolution of 10 m using sentinel-1 and sentinel-2 data on the google earth engine cloud computing platform. **Remote Sensing**, v. 11, n. 1, p. 43, 2019.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, v. 5, n. 4, p. 115-133, 1943.

MELO, D. H. Uso de dados Ikonos II na análise urbana: testes operacionais na zona leste de São Paulo. **São José dos Campos**, 2002.

MENESES, P. R.; ALMEIDA, T. Introdução ao processamento de imagens de sensoriamento remoto. **Universidade de Brasília, Brasília**, 2012.

MUTANGA, O.; KUMAR, L. **Google earth engine applications**. 2019.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. **Sistemas inteligentes-Fundamentos e aplicações**, v. 1, n. 1, p. 32, 2003.

NADIKATTU, R. R. The Supremacy of Artificial intelligence and Neural Networks. **International Journal of Creative Research Thoughts**, v. 5, n. 1, 2017.

NAJAFABADI, M. M. *et al*. Deep learning applications and challenges in big data analytics. **Journal of big data**, v. 2, n. 1, p. 1-21, 2015.

OLOFSSON, P. *et al.* Good practices for estimating area and assessing accuracy of land change. **Remote Sensing of Environment**, v. 148, p. 42-57, 2014.

PANCHAL, G. *et al.* Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers. **International Journal of Computer Theory and Engineering**, v. 3, n. 2, p. 332-337, 2011.

PINTO, L. R. M. **Decisão do espectro baseado em aprendizado de máquina para redes de rádios cognitivos com suporte a comunicação multi-saltos.** 2018. 95 p. Dissertação (Mestrado em Ciência da Computação)–Universidade Federal de Lavras, Lavras, 2018.

RAMACHANDRAN, P.; ZOPH, B.; LE, Q. V. Searching for activation functions. **arXiv preprint arXiv:1710.05941**, 2017.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, v. 65, n. 6, p. 386, 1958.

ROBINSON, N. P. *et al*. A dynamic Landsat derived normalized difference vegetation index (NDVI) product for the conterminous United States. **Remote Sensing**, v. 9, n. 8, p. 863, 2017.

SILVA, J. F. *et al*. Land use/cover (LULC) mapping in brazilian cerrado using neural network with sentinel-2 data. **FLORESTA**, v. 50, n. 3, p. 1430-1438, 2020.

SOUZA, T.; CORREIA, S. Estudo de técnicas de realce de imagens digitais e suas aplicações. In: **II Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica**. 2007. p. 3-10.

SRIVASTAVA, P. K. *et al*. Selection of classification techniques for land use/land cover change investigation**. Advances in Space Research**, v. 50, n. 9, p. 1250-1265, 2012.

TAMIMINIA, H. *et al*. Google Earth Engine for geo-big data applications: A meta-analysis and systematic review. **ISPRS Journal of Photogrammetry and Remote Sensing**, v. 164, p. 152-170, 2020.

VALE, G. B. *et al.* **Reconstrução e reconhecimento de imagens binárias utilizando o algoritmo Máquina de Boltzmann**. 2016.

YANG, J. et al. The role of satellite remote sensing in climate change studies. **Nature climate change**, v. 3, n. 10, p. 875-883, 2013.

APÊNDICES

APÊNDICE A - Código fonte utilizado para criação da Rede Neural

```
from google.colab import auth
auth.authenticate_user()
import ee
ee.Authenticate()
ee.Initialize()
import tensorflow as tf
print(tf.__version__)
import folium
print(folium.__version__)
# Nome de usuário do Google Earth Engine. Usado para importar a imagem
# classificada para a pasta de ativos da plataforma.
USER_NAME = 'danielgacueto'

# Compartimento do Google Cloud Storage em que os dados de treinamento,
# teste e predições erão escritos.
OUTPUT_BUCKET = 'landsatclass'

# Coleção de imagens que será usada no modelo
L8SR = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
# Bandas que serão usadas na predição
BANDS = ['B2_max', 'B3_max', 'B4_max', 'B5_max', 'B6_max', 'B7_max',
         'B2_median', 'B3_median', 'B4_median', 'B5_median', 'B6_median',
'B7_median',
         'B2_stdDev', 'B3_stdDev', 'B4_stdDev', 'B5_stdDev', 'B6_stdDev',
'B7_stdDev',
         'B2_min', 'B3_min', 'B4_min', 'B5_min', 'B6_min', 'B7_min',
         'B2_mean', 'B3_mean', 'B4_mean', 'B5_mean', 'B6_mean', 'B7_mean']

# Localização da base de dados dos pontos cuja classe de uso de solo já está
# definida. Usada para o teste e treinamento do algoritmo.
LABEL_DATA = ee.FeatureCollection('users/danielgacueto/PCTLand4C')
# Os rótulos de cada classe estão contidos nesta colçeção,
# georreferenciados em cada ponto coletado
LABEL = 'landcover'
# Número de rótulos, ou seja, números de classe na classificação.
N_CLASSES = 4

# Esses nomes são usados para especificar propriedades na exportação de
# dados de treinamento / teste e para definir o mapeamento entre nomes e dados
# ao ler conjuntos de dados no formato TensorFlow.
FEATURE_NAMES = list(BANDS)
FEATURE_NAMES.append(LABEL)

# Nome dos arquivos para treinamento e teste.  Estes arquivos em formato
# TFRecord serão exportados do Earth Engine para o Cloud Storage.
```

```
TRAIN_FILE_PREFIX = 'TreinamentoLandsat4C'
TEST_FILE_PREFIX = 'TesteLandsat4C'
file_extension = '.tfrecord.gz'
TRAIN_FILE_PATH = 'gs://' + OUTPUT_BUCKET + '/' + TRAIN_FILE_PREFIX +
file_extension
TEST_FILE_PATH = 'gs://' + OUTPUT_BUCKET + '/' + TEST_FILE_PREFIX + file_extension

# Nome da imagem final usada na predição.  O modelo treinado irá usar
# esta imagem como base e fazer predições em cada pixel da mesma.
IMAGE_FILE_PREFIX = 'Imagem_Landsat4C'

# Caminho de saída da imagem classificada em formato TFRecord.
OUTPUT_IMAGE_FILE = 'gs://' + OUTPUT_BUCKET +
'/Imagem_Classificada_Landsat4C.TFRecord'
# Região para exportação da imagem.
EXPORT_REGION = ee.Geometry.Rectangle([-45.69318650159193, -19.17794777414055,
-44.996927956670056, -18.372471950752168])

# Nome do arquivo a ser criado com a importação da imagem classificada
# no formato TFRecord para o Cloud Storage
OUTPUT_ASSET_ID = 'users/' + USER_NAME + '/Imagem_Classificada_Landsat4C'
# Função de máscara de nuvem
def maskL8sr(image):
  cloudShadowBitMask = ee.Number(2).pow(3).int()
  cloudsBitMask = ee.Number(2).pow(5).int()
  qa = image.select('pixel_qa')
  mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0).And(
    qa.bitwiseAnd(cloudsBitMask).eq(0))
  return
image.updateMask(mask).select('B1','B2','B3','B4','B5','B6','B7').divide(10000)

# A imagem de entrada é um composto das estatísticas de bandas
# das imagens do ano de 2020 com a máscara de nuvens já aplicada
coll1 = L8SR.filterDate('2020-01-01',
'2021-01-01').map(maskL8sr).filterBounds(EXPORT_REGION)
image =
coll1.reduce(ee.Reducer.minMax().combine(reducer2=ee.Reducer.stdDev(),sharedInputs=
True)
.combine(reducer2=ee.Reducer.median(),sharedInputs=True).combine(reducer2=ee.Reduce
r.mean(),sharedInputs=True))

# Use a biblioteca folium para visualizar a imagem formada e certificar que o
processo ocorreu corretamente.
mapid = image.getMapId({'bands': ['B4_median', 'B3_median', 'B2_median'], 'min': 0,
'max': 0.3})
folium.TileLayer(
    tiles=mapid['tile_fetcher'].url_format,
    attr='Map Data &copy; <a href="https://earthengine.google.com/">Google Earth
Engine</a>',
```

```
    overlay=True,
    name='median composite',
  ).add_to(map)
map.add_child(folium.LayerControl())
map
# Associar os dados coletados à imagem
sample = image.sampleRegions(
  collection=LABEL_DATA, properties=[LABEL], scale=30).randomColumn()

# Repartir as amostras na proporção 70-30.
training = sample.filter(ee.Filter.lt('random', 0.7))
testing = sample.filter(ee.Filter.gte('random', 0.7))

from pprint import pprint

#Imprimir os primeiros pontos para verificar se o procedimento funcionou.
pprint({'training': training.first().getInfo()})
pprint({'testing': testing.first().getInfo()})
O bucket deve ser controlado pelo usuário, que deve poder ler e escrever no mesmo.
print('Found Cloud Storage bucket.' if tf.io.gfile.exists('gs://' + OUTPUT_BUCKET)
    else 'Can not find output Cloud Storage bucket.')
# Criando as tarefas de exportação
training_task = ee.batch.Export.table.toCloudStorage(
  collection=training,
  description='Training Export',
  fileNamePrefix=TRAIN_FILE_PREFIX,
  bucket=OUTPUT_BUCKET,
  fileFormat='TFRecord',
  selectors=FEATURE_NAMES)

testing_task = ee.batch.Export.table.toCloudStorage(
  collection=testing,
  description='Testing Export',
  fileNamePrefix=TEST_FILE_PREFIX,
  bucket=OUTPUT_BUCKET,
  fileFormat='TFRecord',
  selectors=FEATURE_NAMES)
# Iniciando as tarefas
training_task.start()
testing_task.start()
# Mostra as tarefas em andamento
pprint(ee.batch.Task.list())
# Exportação da imagem
image_export_options = {
  'patchDimensions': [256, 256],
  'maxFileSize': 104857600,
  'compressed': True
}
```

```
# Configurando a tarefa de exportação
image_task = ee.batch.Export.image.toCloudStorage(
  image=image,
  description='Image Export',
  fileNamePrefix=IMAGE_FILE_PREFIX,
  bucket=OUTPUT_BUCKET,
  scale=30,
  fileFormat='TFRecord',
  region=EXPORT_REGION.toGeoJSON()['coordinates'],
  formatOptions=image_export_options,
)
# Inicia as tarefas
image_task.start()
# Mostra as tarefas em andamento
pprint(ee.batch.Task.list())
import time

while image_task.active():
  print('Polling for task (id: {}).'.format(image_task.id))
  time.sleep(30)
print('Done with image export.')
# Criar um dataset a partir dos dados TFRecord exportados.
train_dataset = tf.data.TFRecordDataset(TRAIN_FILE_PATH, compression_type='GZIP')
# Imprimir os primeiros registros para conferir se estão certos.
print(iter(train_dataset).next())
columns = [
  tf.io.FixedLenFeature(shape=[1], dtype=tf.float32) for k in FEATURE_NAMES
]
features_dict = dict(zip(FEATURE_NAMES, columns))

pprint(features_dict)
def normalized_difference(a, b):

  nd = (a - b) / (a + b)
  nd_inf = (a - b) / (a + b + 0.000001)
  return tf.where(tf.math.is_finite(nd), nd, nd_inf)

def add_NDVI(features, label):

  features['NDVI'] = normalized_difference(features['B5'], features['B4'])
  return features, label
def parse_tfrecord(example_proto):
  parsed_features = tf.io.parse_single_example(example_proto, features_dict)
  labels = parsed_features.pop(LABEL)
  return parsed_features, tf.cast(labels, tf.int32)

# Mapear a função em todo o dataset.
parsed_dataset = train_dataset.map(parse_tfrecord, num_parallel_calls=5)
```

```
# Imprimir os primeiros registros analisados para conferência.
pprint(iter(parsed_dataset).next())
from tensorflow import keras

# Adicionar o NDVI criado à base de dados.
input_dataset = parsed_dataset.map(add_NDVI)

# Keras requer a entrada de dados em formato énuplo.
# Para a função de perda de entropia cruzada categorizada
# é necessário que os rótulos sejam transormados em um vetor one-hot.
def to_tuple(inputs, label):
  return (tf.transpose(list(inputs.values())),
          tf.one_hot(indices=label, depth=N_CLASSES))

input_dataset = input_dataset.map(to_tuple).batch(8)

# Definir as variáveis no modelo
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(64, activation=tf.nn.relu),
  tf.keras.layers.Dense(64, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(N_CLASSES, activation=tf.nn.softmax)
])

# Compilação do modelo com a função de perda escolhida.
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Ajuste do modelo à base de dados.
model.fit(x=input_dataset, epochs=80)
test_dataset = (
  tf.data.TFRecordDataset(TEST_FILE_PATH, compression_type='GZIP')
    .map(parse_tfrecord, num_parallel_calls=5)
    .map(add_NDVI)
    .map(to_tuple)
    .batch(1))

model.evaluate(test_dataset)
# Lista todos os arquivos presentes no bucket diretório.
files_list = !gsutil ls 'gs://'{OUTPUT_BUCKET}
# Seleciona apenas arquivos gerados na função de exportação das imagens
exported_files_list = [s for s in files_list if IMAGE_FILE_PREFIX in s]

# Lista todos arquivos com seus respectivos anexos JSON.
image_files_list = []
json_file = None
for f in exported_files_list:
  if f.endswith('.tfrecord.gz'):
```

```
    image_files_list.append(f)
  elif f.endswith('.json'):
    json_file = f

# Coloca os arquivos na ordem correta.
image_files_list.sort()

pprint(image_files_list)
print(json_file)
import json
# Anexa a metadata dos arquivos JSON
json_text = !gsutil cat {json_file}
mixer = json.loads(json_text.nlstr)
pprint(mixer)
# Extrai informações do arquivo JSON.
patch_width = mixer['patchDimensions'][0]
patch_height = mixer['patchDimensions'][1]
patches = mixer['totalPatches']
patch_dimensions_flat = [patch_width * patch_height, 1]

# Os tensores estão em sequência, uma sequência por banda da imagem.
image_columns = [
  tf.io.FixedLenFeature(shape=patch_dimensions_flat, dtype=tf.float32)
    for k in BANDS
]

# Chaves da função de análise
image_features_dict = dict(zip(BANDS, image_columns))

# É possível fazer um dataset a partir de várias imagens especificadas na lista
image_dataset = tf.data.TFRecordDataset(image_files_list, compression_type='GZIP')

# Função de análise
def parse_image(example_proto):
  return tf.io.parse_single_example(example_proto, image_features_dict)

# Transformar o dataset em um longo tensor por sequência de dados
image_dataset = image_dataset.map(parse_image, num_parallel_calls=5)

# Quebrar o tensor longo em vários menores.
image_dataset = image_dataset.flat_map(
  lambda features: tf.data.Dataset.from_tensor_slices(features)
)

# Adicionar o NDVI
image_dataset = image_dataset.map(
  # NDVI deve ser atrelado à uma função ainda sem rótulo.
  lambda features: add_NDVI(features, None)[0]
)
```

```
# Transforma as chaves da função de análise em uma sequência énupla.
image_dataset = image_dataset.map(
  lambda data_dict: (tf.transpose(list(data_dict.values())), )
)

# Transforma cada sequência em um lote de processamento
image_dataset = image_dataset.batch(patch_width * patch_height)
# Executa a predição em lotes, usando o número de passos igual ao número de lotes.
predictions = model.predict(image_dataset, steps=patches, verbose=1)

# As predições são geradas em forma de matriz.
print(predictions[0])
# Iniciar a transformação.
writer = tf.io.TFRecordWriter(OUTPUT_IMAGE_FILE)

# Cada sequência de predições irá adicionar uma feição com as predições geradas
# no arquivo de saída. Como as predições já foram oordenadas em sequência
anteriormente,
# as sequências criadas aqui já estarão na ordem certa
patch = [[], [], [], []]
cur_patch = 1
for prediction in predictions:
  patch[0].append(tf.argmax(prediction, 1))
  patch[1].append(prediction[0][0])
  patch[2].append(prediction[0][1])
  patch[3].append(prediction[0][2])
  # Assim que uma sequência de classes finalizar...
  if (len(patch[0]) == patch_width * patch_height):
    print('Done with patch ' + str(cur_patch) + ' of ' + str(patches) + '...')
    # Cria-se um exemplo
    example = tf.train.Example(
      features=tf.train.Features(
        feature={
          'prediction': tf.train.Feature(
              int64_list=tf.train.Int64List(
                  value=patch[0])),
          'bareProb': tf.train.Feature(
              float_list=tf.train.FloatList(
                  value=patch[1])),
          'vegProb': tf.train.Feature(
              float_list=tf.train.FloatList(
                  value=patch[2])),
          'waterProb': tf.train.Feature(
              float_list=tf.train.FloatList(
                  value=patch[3])),
        }
      )
    )
```

```
    # Os exemplos são registrados no arquivo de saída as sequências são
reorganizadas
    # para mais um lote ser escrito.
    writer.write(example.SerializeToString())
    patch = [[], [], [], []]
    cur_patch += 1

writer.close()
!gsutil ls -l {OUTPUT_IMAGE_FILE}
print('Uploading to ' + OUTPUT_ASSET_ID)
# Iniciar o upload.
!earthengine upload image --asset_id={OUTPUT_ASSET_ID} --pyramiding_policy=mode
{OUTPUT_IMAGE_FILE} {json_file}
```

APÊNDICE B - Código fonte para coleta de dados de treinamento e validação

```
function maskL8sr(image) {
  // Bits 3 and 5 are cloud shadow and cloud, respectively.
  var cloudShadowBitMask = (1 << 3);
  var cloudsBitMask = (1 << 5);
  // Get the pixel QA band.
  var qa = image.select('pixel_qa');
  // Both flags should be set to zero, indicating clear conditions.
  var mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0)
                 .and(qa.bitwiseAnd(cloudsBitMask).eq(0));
  return image.updateMask(mask);
}

var dataset = ee.ImageCollection(2020-01-01', '2020-12-31')
                  .filterBounds(regiao3)
                  .map(maskL8sr);

var fullimg = dataset.reduce(ee.Reducer.mean().combine({
  reducer2: ee.Reducer.stdDev(),
  sharedInputs: true
}).combine({
  reducer2: ee.Reducer.minMax(),
  sharedInputs: true
}).combine({
  reducer2: ee.Reducer.median(),
  sharedInputs: true}
));

var visualization = {
  min: 0.0,
  max: 3000,
  bands: ['B5_median', 'B4_median', 'B3_median'],
};

Map.addLayer(fullimg, visualization, 'RGB');
## Coletar os pontos de treinamento na plataforma e salvá-los com o formato
## FeatureCollection, cada classe com um número inteiro distinto.
var colecaoPCTLand4c = agua.merge(vegnativa).merge(solourbano).merge(agri);

Export.table.toAsset({
  collection: colecaoPCTLand4c,
 description: 'PCTLand4C',
 assetId: 'PCTLand4C'
})
```