UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO CENTRO TECNOLÓGICO DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROJETO DE GRADUAÇÃO

Luma Figueiredo de Oliveira Pontes

DETECÇÃO DE ÁREAS SEM CONSTRUÇÕES CIVIS EM MEIO À MALHA URBANA UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

VITÓRIA - ES

2021

Luma Figueiredo de Oliveira Pontes

DETECÇÃO DE ÁREAS SEM CONSTRUÇÕES CIVIS EM MEIO À MALHA URBANA UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

Parte manuscrita do Projeto de Graduação da aluna Luma Figueiredo de Oliveira Pontes, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Patrick Marques Ciarelli

VITÓRIA - ES

2021

Luma Figueiredo de Oliveira Pontes

DETECÇÃO DE ÁREAS SEM CONSTRUÇÕES CIVIS EM MEIO À MALHA URBANA UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

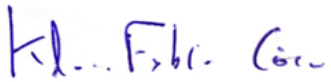
Parte manuscrita do Projeto de Graduação da aluna Luma Figueiredo de Oliveira Pontes, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovada em 5 de outubro de 2021.

COMISSÃO EXAMINADORA:

Prof. Dr. Patrick Marques Ciarelli
Universidade Federal do Espírito Santo
Orientador

MSc. Ana Paula Miranda Diniz
Examinadora



Prof. Dr. Klaus Fabian Côco
Universidade Federal do Espírito Santo
Examinador

VITÓRIA - ES

2021

RESUMO

Existe uma preocupação governamental constante com habitações formadas em zonas impróprias para moradia. Essa prática é ocasionada por diversas questões históricas e sociais e acarreta um maior gasto público e pior qualidade de vida para a população que se encontra nessa condição. Uma solução para esta questão é inserir em áreas urbanas as famílias que vivem em situações impróprias, onde o acesso a diversos serviços públicos é mais fácil. Essa inserção deve ser feita através da construção de moradias populares em vazios urbanos classificados como Zonas Especiais de Interesse Social (ZEIS). Este trabalho propõe uma rede de aprendizado profundo que pode ser usado para auxiliar o processo de localização de ZEIS na Grande Vitória a fim de otimizar essa tarefa. Com este objetivo em mente, foram obtidas imagens de satélite de áreas urbanas das cidades de Vila Velha, Vitória e Serra no Estado do Espírito Santo por meio da API Google Maps. As imagens obtidas foram classificadas e pré-processadas para compor uma base de dados usada no treino e avaliação de uma rede neural convolucional para a identificação de ZEIS em meio à malha urbana. Uma análise do desempenho da rede foi feita com o uso das técnicas de aumento de dados e *transfer learning* com diferentes redes, sendo estas a Resnet 50, Resnet 101 e Resnet 152. O resultado experimental mais satisfatório foi obtido pelo modelo treinado com aumento de dados e com a Resnet 50, com um *F1-score* de 93,51%.

Palavras-chave: Vazios urbanos; Aprendizado profundo; Rede neural convolucional; *Transfer Learning*.

ABSTRACT

There is a constant government concern with housings formed in unsuitable areas for dwelling. This practice is caused by several historical and social issues and entails greater public spending and low life quality for the population in this condition. One solution to this issue is to place families living in inappropriate situations in urban areas, where access to public services is easier. This placing must be done through the construction of social housing in urban vacant land classified as Special Areas Of Social Interest (Zonas Especiais de Interesse Social – ZEIS). This work proposes a deep learning network that can help in the process of locating ZEIS at Grande Vitória in order to optimize this task. With this goal in mind, satellite images of urban areas in the cities of Vila Velha, Vitória, and Serra in the State of Espírito Santo were obtained through the API Google Maps. The images obtained were classified and pre-processed to compose a dataset used in the training and evaluation of a convolutional neural network for the identification of ZEIS in urban areas. An analysis of the network performance was carried out using data augmentation and transfer learning techniques with different networks: Resnet 50, Resnet 101, and Resnet 152. The most satisfactory experimental result was achieved by the model trained with data augmentation and with Resnet 50 with a F1-score of 93,51%.

Keywords: Urban Vacant Land; Deep Learning; Convolutional Neural Network; Transfer Learning.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CNN	*Convolutional Neural Networks*
CPU	*Central Process Unit*
DL	*Deep Learning*
FN	*False Negative*
FP	*False Positive*
GPU	*Graphics Processing Unit*
IBGE	Instituto Brasileiro de Geografia e Estatística
IJSN	Instituto Jones Santos Neves
JIT	*Just in Time*
PMV	Prefeitura Municipal de Vitória
PNG	*Portable Network Graphics*
ReLU	*Rectified Linear Unit*
Resnet	*Residual Network*
RMGV	Região Metropolitana da Grande Vitória
TN	*True Negative*
TP	*True Positive*
UFES	Universidade Federal do Espírito Santo
ZEIS	Zonas Especiais de Interesse Social

SUMÁRIO

1 INTRODUÇÃO

1.1 Apresentação

No Brasil, assim como em outros países em desenvolvimento, nota-se um grande déficit habitacional (FUNDAÇÃO JOÃO PINHEIRO, 2021) e, em geral, isso acontece pela falta de acesso à terra, seja por seu preço, pela especulação imobiliária entre outros. Como consequência, pessoas de origem mais humilde buscam zonas periféricas aos centros urbanos para residir, onde, na maioria das vezes, não há acesso à saneamento básico, serviços urbanos, saúde pública e educação, criando áreas de maior complexidade social nas cidades.

Contudo, quando se analisa as áreas urbanas, percebe-se muitas zonas ociosas, sendo estas propriedades urbanas, que não cumprem sua função social (BRASIL, 2001). Esses espaços, comumente chamados de vazios urbanos, estão localizados nas regiões consolidadas das cidades que já possuem acesso ao saneamento básico, serviços públicos, vias pavimentadas, ou seja, uma infraestrutura capaz de absorver o crescimento populacional dos centros urbanos.

O espraiamento das cidades onera a gestão pública porque aumenta o custo de manutenção da infraestrutura urbana como por exemplo o custo de construções de mais unidades de saúde pública, escolas e creches públicas, entre outros, para que toda população tenha fácil acesso a esses serviços. É mencionada no Programa de Metas da Prefeitura Municipal de Vitória (PMV) para o período de 2021 a 2024 (PREFEITURA MUNICIPAL DE VITÓRIA, 2020) a preocupação com o tópico.

O Plano de Desenvolvimento Urbano Integrado (PDUI), da Região Metropolitana da Grande Vitória (RMGV), é uma das metas do planejamento municipal previsto na Lei Orgânica do Município de Vitória que, entre outros objetivos de desenvolvimento urbano, trata da importância do aproveitamento de infraestruturas existentes nas malhas urbanas (ESPÍRITO SANTO, 2017). Por meio de um melhor planejamento governamental, é possível induzir a utilização dos vazios em meio às cidades, buscando o que é chamado de cidade compacta (FRAGMAQ, 2014).

Visando a melhor utilização das áreas ociosas das cidades, este projeto tem como objetivo a construção de uma aplicação para o mapeamento dos vazios urbanos na RMGV. Para tal, será implementado um mecanismo de visão computacional aplicado a imagens de satélites baseado em redes neurais convolucionais. O método a ser desenvolvido neste estudo poderá subsidiar o poder público a tomada de decisão no planejamento urbano, como construção de habitação de interesse social ou alocação de novos equipamentos públicos que melhorem a qualidade de vida da população.

1.2 Relevância para Gestão Pública

O interesse em realizar a melhor ocupação das cidades está presente em vários estados (COSTA; CHAVES, 2019). No Espírito Santo não é diferente, e o Instituto Jones Santos Neves (IJSN), que é uma autarquia vinculada à Secretaria do Estado de Economia e Planejamento, é incumbido da organização da base de dados estatísticos e georreferenciados do Estado. Nele são dirigidos estudos e pesquisas objetivando fomentar o desenvolvimento socioeconômico do Espírito Santo e gerar informações atualizadas sobre o Estado (INSTITUTO JONES DOS SANTOS NEVES, 2019). A Coordenação de Estudos Territoriais do IJSN, atualmente, vem buscando promover pesquisas inovadoras em parceria com instituições de ensino que auxiliem no enfrentamento de problemas urbanos do Estado.

O trabalho proposto para este projeto derivou-se da iniciativa do IJSN. A Coordenação de Estudos Territoriais do Instituto, visando impulsionar o planejamento urbano das cidades do Espírito Santo, tem como objetivo inovar na forma de coleta e sistematização de informações territoriais (INSTITUTO JONES DOS SANTOS NEVES, 2019). É necessário, para acompanhar o dinamismo da transformação das cidades na atualidade, a implementação de um sistema de coleta de dados que possa encurtar a espera de atualização de dados censitários feitos em pesquisas tradicionais, como aquelas realizadas pelo Instituto Brasileiro de Geografia e Estatística (IBGE).

O Governo do Espírito Santo tem dado prioridade a projetos de inovação nos últimos anos, vide o programa ES Inovador e o Manifesto da Inovação (UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO, 2019). A iniciativa de incentivo à inovação do Governo vai ao encontro das demandas de interesse do IJSN.

Para mais, o Programa de Metas de 2021-2024, da PMV (PREFEITURA MUNICIPAL DE VITÓRIA, 2020), trata do tópico de Zonas Especiais de Interesse Social (ZEIS) na Meta 56, onde coloca como objetivo “ampliar a concessão de moradias definitivas para famílias inseridas em áreas impróprias para habitação”. Essa meta pode ser vista como uma continuação da Meta 19 do Programa de metas até 2020 (PREFEITURA MUNICIPAL DE VITÓRIA, 2017) que ambicionava “aumentar o número de novas escrituras em áreas de interesse social entregues”. Na justificativa do projeto abrangido por ambas as metas é ressaltada a condição de famílias de baixa renda que habitam em áreas ocupadas irregularmente.

O Governo Federal, com a mesma preocupação em mente, deu início ao programa Minha Casa Minha Vida em 2009 para facilitar o acesso à moradia no Brasil. O programa subsidia e oferece juros menores no financiamento do imóvel para famílias de baixa renda (ABE, 2019).

Isso destaca o modo como os territórios têm sido ocupados. A dificuldade ao acesso à habitação nos centros urbanos, que prejudica a população de baixa renda, cria a necessidade de implementação de projetos inovadores para solucionar problemas urbanos e de buscar ferramentas de mapeamento e gestão do território que permitam analisar de forma ágil e efetiva o cenário urbano atual.

1.3 Objetivos

1.3.1 Objetivo Geral

O projeto proposto tem como objetivo geral a implementação de um sistema que auxilie a identificação de vazios urbanos, aqui denominados de ZEIS, por meio de análise de imagens de satélite na área da Grande Vitória. Para realizar esta tarefa, pretende-se usar redes neurais convolucionais treinadas com imagens de satélite

das regiões urbanas das cidades de Vitória, Vila Velha e Serra do Estado do Espírito Santo.

1.3.2 Objetivos Específicos

Para alcançar o objetivo geral, foram estipulados os seguintes objetivos específicos:

I. Elaborar o algoritmo de coleta de imagens de satélite do Google Maps, utilizando a *Application Programming Interface* (API) criada e disponibilizada pela Google;
II. Coletar e rotular as imagens para compor a base de dados para teste e treino da rede desenvolvida;
III. Definir uma arquitetura de rede neural convolucional a ser usada;
IV. Treinar e avaliar o desempenho da rede treinada.

1.4 Estrutura do texto

O presente trabalho está estruturado de modo que o capítulo 1 apresenta uma introdução à problemática que o trabalho se propõe a solucionar, a sua relevância e a estratégia básica de solução pretendida, e os objetivos da realização deste projeto. O capítulo 2 apresenta conceitos fundamentais para entendimento da solução proposta.

Posteriormente, no capítulo 3, o texto é dedicado à solução proposta, sendo detalhadas as etapas do projeto construído para a resolução do problema abordado. Os resultados obtidos são apresentados no capítulo 4. Finalmente, no capítulo 5, são apresentadas as conclusões relativas ao trabalho e possíveis caminhos futuros.

2 EMBASAMENTO TEÓRICO

2.1 Introdução

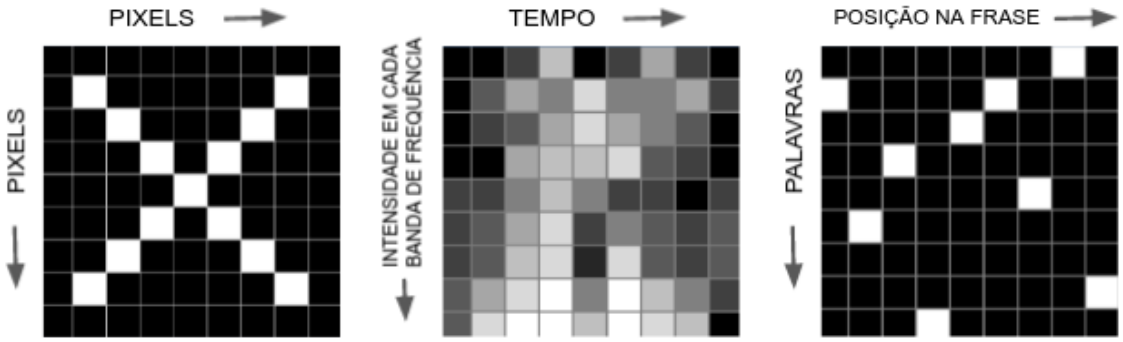
A fim de estabelecer uma base para o entendimento do trabalho, esse capítulo expõe alguns conceitos teóricos relevantes para o projeto. Primeiramente, a definição e funcionamento de uma rede neural convolucional são explicados, em seguida, são definidos o conceito de *transfer learning* e a arquitetura das redes ResNet (que serão usadas neste trabalho). Por fim, é apresentada uma visão geral da Interface de Programação de Aplicação (API, do inglês *Application Programming Interface*) Google Maps usada no desenvolvimento do projeto.

2.2 Rede Neural Convolucional

Uma Rede Neural Convolucional (CNN, do inglês *Convolutional Neural Network*) é um tipo de Rede Neural Artificial que usa a operação de convolução em pelo menos uma de suas etapas. Os dados de entrada de uma rede de arquitetura CNN têm formato de matrizes ou vetores, por isso esse tipo de rede é muito usado em tarefas que usam imagens, que podem ser entendidas como matrizes, nas quais o menor elemento é chamado de pixel. Essa categoria de rede também pode ser usada para reconhecer padrões em textos e áudios, pois tais informações também podem ser representadas por matrizes como mostra a Figura 1 (ROHRER, 2010).

É importante destacar que uma CNN identifica padrões locais espaciais nas matrizes, isto é, a organização espacial dos valores da matriz deve estar ligada ao seu significado. Isso quer dizer que “se seu dado continua sendo útil depois de trocar as colunas de ordem, então não é possível usar uma CNN” (ROHRER, 2010, p. 265, tradução nossa).

Figura 1 – Representação gráfica de 3 tipos de dados que podem ser usados como entrada de uma CNN: imagem (à esquerda), áudio (no centro) e texto (à direita)

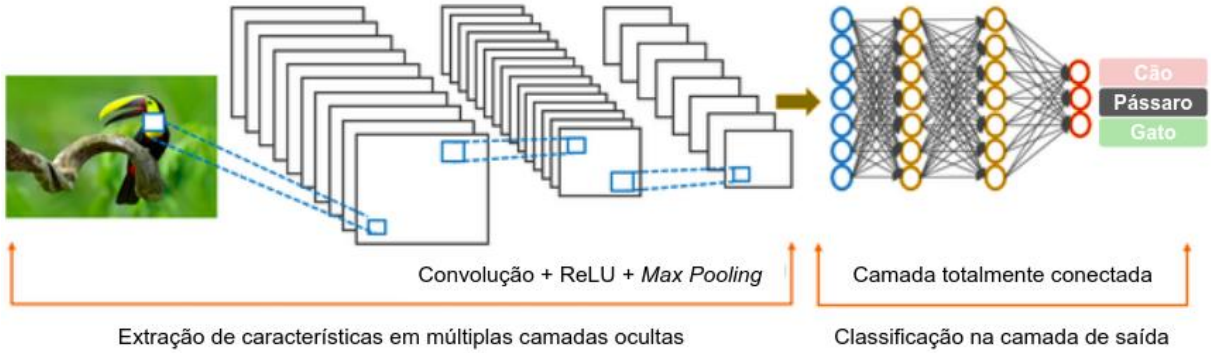


Fonte: Rohrer (2010).
Nota: Modificada pela autora.

A CNN é comumente utilizada como uma arquitetura de aprendizado profundo (DL, do inglês *Deep Learning*). O DL é um subcampo de aprendizagem de máquina que pode ser definido como uma recaracterização das redes neurais com múltiplas camadas ocultas (GOMES, 2014). Este método tem como objetivo modelar relações complexas entre os dados estudados que arquiteturas clássicas de redes neurais não conseguem realizar de forma satisfatória.

A implementação básica de uma CNN funciona de forma a processar os dados de entrada em vários níveis de representação, chamados também de camadas, sendo a saída do processamento de uma camada, a entrada da próxima (DENG; YU, 2014). Na Figura 2 é apresentado graficamente uma rede de aprendizado profundo do tipo CNN.

Figura 2 – Representação gráfica de uma Rede Neural Convolucional



Fonte: Techwasti (2019).

Nota: Traduzido pela autora.

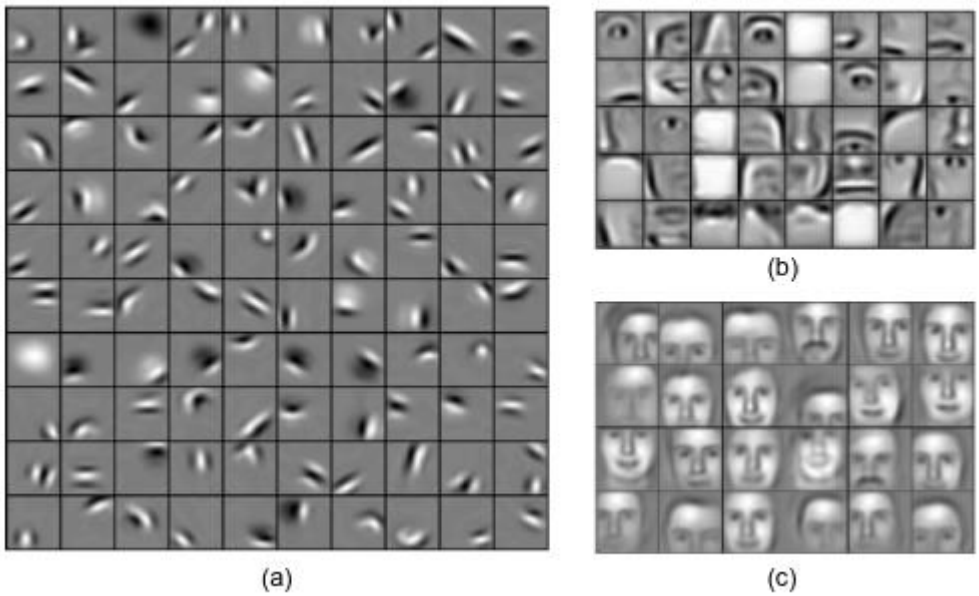
Na estrutura da CNN há uma camada de entrada, que é composta por dados coletados do objeto de estudo, e uma camada de saída composta pelos resultados das predições feitas pela rede. No caso deste trabalho, a entrada serão imagens de satélite e a saída será a predição da rede se existem ou não regiões de ZEIS nas imagens. Subsequente à camada de entrada e anterior à camada de saída existem múltiplas camadas ocultas.

A parte inicial da CNN é composta por camadas de extração de características, onde várias operações são feitas sobre a matriz de entrada. Quanto maior o número de camadas, possivelmente características mais rebuscadas são extraídas. Para ilustrar melhor a relação entre o número de camadas do processo de extração de características e a complexidade dos elementos a serem identificados nas imagens, primeiro deve-se entender que as características de uma imagem podem ser descritas em vários níveis de complexidade, por exemplo, colocando de forma crescente, a intensidade dos *pixels*, bordas, partes de objetos, objetos e assim por diante (LEE; GROSSE; RANGANATH; NG, 2009).

Tomando como exemplo uma CNN que é treinada para identificação de faces, nas primeiras camadas da rede haveria a identificação de bordas de diferentes ângulos. Nas próximas camadas o algoritmo seria capaz de identificar partes de um rosto, como narizes, olhos e bocas. Já nas últimas camadas, as faces já seriam identificadas por inteiro. Pode-se ver o resultado das camadas da extração de características na Figura 3, retirada do trabalho (LEE et al., 2009), em que a camada

atual combina elementos da camada anterior para formar estruturas mais complexas.

Figura 3 – Características com diferentes níveis de complexidade como (a) a identificação de bordas, (b) a identificação de narizes, olhos e bocas e (c) identificação de faces



Fonte: Lee, Grosse, Ranganath e Ng (2009).

Posterior às camadas de extração de características, o processo de uma CNN envolve uma série de camadas totalmente conectadas. As conexões ponderadas são estabelecidas entre essas camadas por meio dos resultados de processamento dos dados, de modo que, a saída de uma camada é a entrada da próxima camada no sentido do fluxo de processamento. A última camada totalmente conectada é responsável pela classificação final dos dados de entrada.

2.2.1 Convolução

Ao comparar duas imagens, apesar de não idênticas, humanos conseguem identificar características similares de maneira simples. Já essa mesma comparação, sendo feita matematicamente por computadores, não é trivial. Para fazer isto é necessário um processo chamado de extração de características, que nas redes

convolucionais é feito através de uma série de camadas de convolução, funções de transferência e camadas de *pooling* (NAVAMANI, 2019).

A característica que diferencia uma CNN das demais redes neurais profundas é a presença de camadas convolucionais em sua arquitetura. Nesta camada uma operação de convolução é feita entre a matriz de entrada e uma máscara quadrada ou retangular, também chamada de *kernel*. A operação de convolução é descrita matematicamente na Equação 1, onde $\boldsymbol{f}$ é a matriz de entrada e $\boldsymbol{w}$ é um *kernel* com dimensões $(2\boldsymbol{a}+1) \times (2\boldsymbol{b}+1)$ (KUNDUR, 2015).

$$w(x,y) \circ f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{1}$$

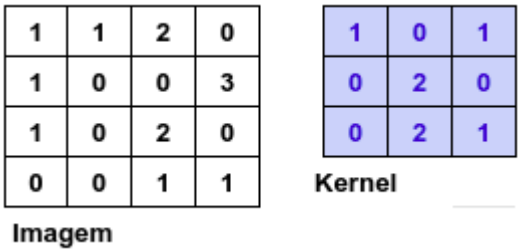
Para entendimento da operação de convolução de forma gráfica pode-se observar a Figura 4. É feita uma varredura na matriz de entrada com o *kernel*, de modo que em cada iteração o *kernel* é centralizado sobre um pixel (elemento da matriz) e é feita a soma das multiplicações ponto a ponto dos elementos do *kernel* pelos elementos da matriz que estão sob o *kernel*. O retorno da convolução, que acontece após todas as iterações, é uma matriz chamada de mapa de características (DERTAT, 2017b).

2.2.2 *Pooling*

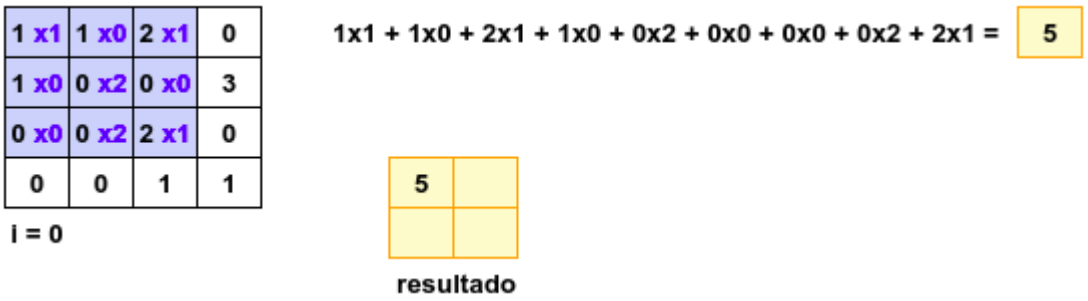
A camada de *pooling* é responsável por diminuir o tamanho dos mapas de características. Esse procedimento combate o sobreajuste (*overfitting*) da rede e ainda diminui o tempo de treino (DERTAT, 2017b). Há diversas técnicas de *pooling* disponíveis, como *mean-pooling*, *average-pooling*, *stochastic-pooling* (GHOLAMALINEZHAD; KHOSRAVI, 2020), entre outras. O mais comumente usado, no entanto, é o *max-pooling*.

A técnica de *max-pooling* consiste em varrer a matriz de entrada com uma janela de tamanho predefinido, iterando com passos (*strides*) também predefinidos. A cada iteração é selecionado o maior valor da região da matriz de entrada que está abaixo da janela e com ele é montada uma nova matriz. Pode-se ver esse processo na Figura 5, onde o *stride* é igual a 2, na horizontal e vertical, e a janela de *pooling* tem tamanho 2×2. O resultado do *max-pooling* pode ser observado na Figura 6.

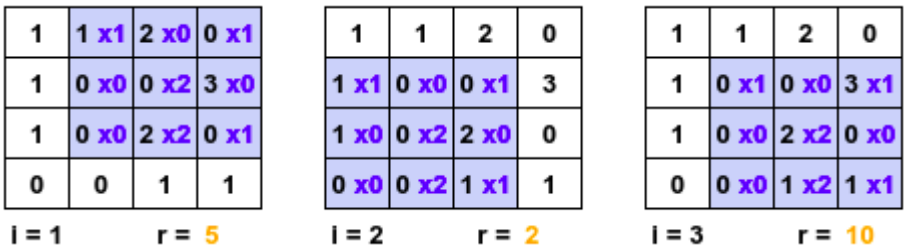
Figura 4 – Etapas da convolução



(a) Imagem de entrada e *kernel* usados na convolução.



(b) Primeira iteração da convolução.

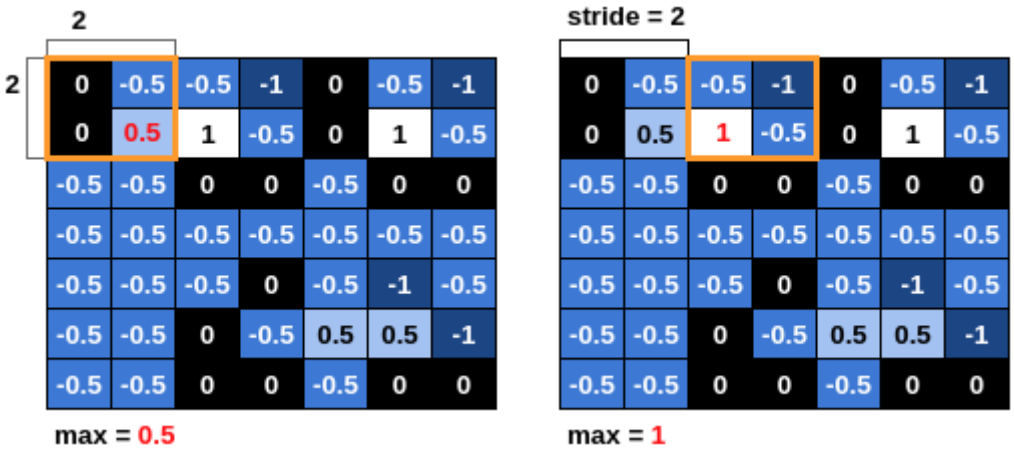


(c) Iterações subsequentes da convolução.

(d) Resultado da convolução.

Fonte: Produção da própria autora.

Figura 5 – Processo de *max-pooling*



Fonte: Produção da própria autora.

Figura 6 – Resultado do *max-pooling* da figura 5

Fonte: Produção da própria autora.

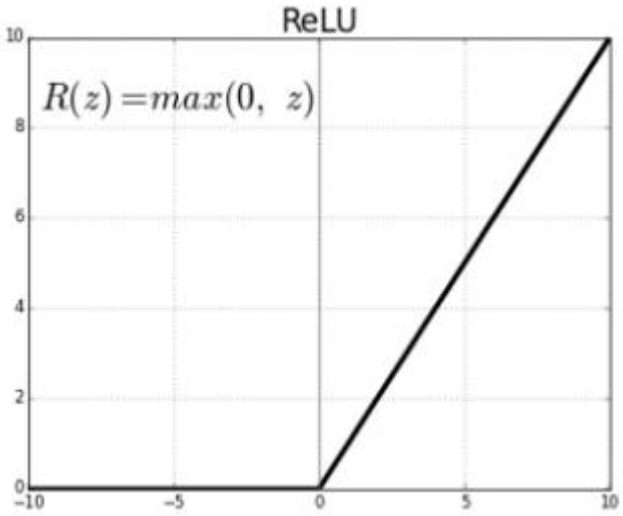
2.2.3 Funções de Ativação

A função de ativação é aplicada nas saídas de uma camada de convolução para ajustar os valores a serem usados como entrada da próxima camada. A relevância da função de ativação é de adicionar não linearidades na rede (JAIN, 2019). Sem isso, ela seria composta somente por operações de soma e multiplicação, ou seja, sempre buscando uma aproximação linear para a obtenção da superfície discriminante.

Existem vários tipos de funções de ativação. A mais usada com a CNN é a função

linear retificada (ReLU, do inglês *rectified linear unit*) (SHARMA, 2017). Essa função transforma todos os valores negativos em zero, como ilustrado no gráfico da Figura 7.

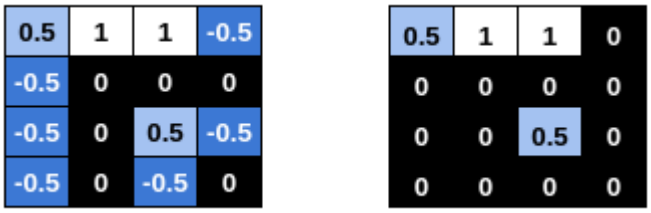
Figura 7 – Gráfico da função ReLU



Fonte: Towards Data Science (2017).

A figura 8 mostra a aplicação da função de ativação ReLU na matriz da figura 6.

Figura 8 – Matriz antes e depois da aplicação do ReLU



Fonte: Produção da própria autora.

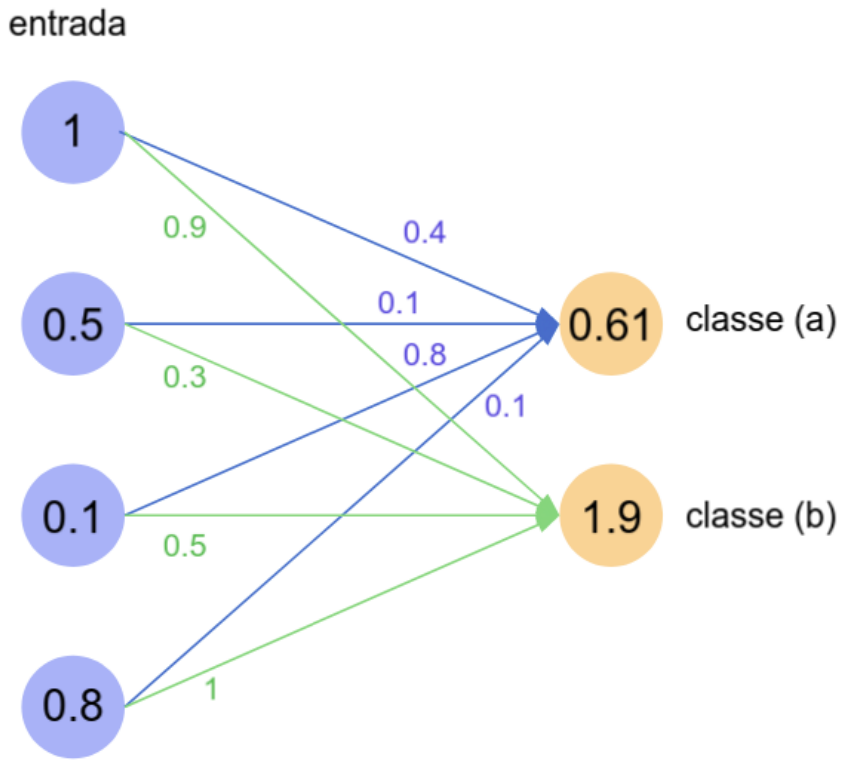
2.2.4 Camadas Totalmente Conectadas

As camadas totalmente conectadas integram a parte final da rede convolucional. Como o nome sugere, todos os elementos de entrada da camada estão conectados a todos os seus neurônios (DERTAT, 2017a), conforme representado na Figura 9, onde as quatro entradas estão conectadas aos dois neurônios. Os elementos de entrada são os mapas de características obtidos na última camada convolucional da rede (seguida pela função de ativação) após o processo de *flatten* (achatamento)

dos dados, no qual as matrizes são convertidas em vetores (DERTAT, 2017a). Os neurônios de saída desta camada são obtidos após uma soma ponderada das entradas pelos pesos das conexões, seguida pela aplicação de uma função de ativação, que normalmente é uma função não linear (DERTAT, 2017a). Nesse caso, também pode ser usada a função ReLU, além de várias outras, como a sigmóide, tangente hiperbólica, *softmax* e assim por diante (DERTAT, 2017a).

Uma rede pode ter uma ou mais camadas totalmente conectadas, sendo a última camada em uma rede de classificação é a responsável por fornecer o rótulo à entrada da rede a partir das características extraídas e processamentos realizados ao longo da rede.

Figura 9 – Camada totalmente conectada com quatro entradas e dois neurônios



Fonte: Produção da própria autora.

2.2.5 Treinamento

As conexões entre as camadas totalmente conectadas e os *kernels* das camadas convolucionais possuem pesos que guiam a resposta da classificação. O treino de

uma rede consiste em ajustar os valores desses pesos com o propósito de melhorar a qualidade de resposta da rede ao resolver uma tarefa específica.

Essa otimização dos pesos ocorre, de forma simplificada, da seguinte maneira: são definidos pesos iniciais aleatórios para a rede e, a partir dos mesmos, ocorre a primeira iteração de treino da rede. Em seguida, com a base de dados usada para o treino, é calculado o quanto as respostas obtidas na iteração divergem das respostas esperadas. Este valor é chamado de erro da rede. Assim, a rede é percorrida no sentido contrário (da saída para a entrada), ajustando ligeiramente os pesos, de modo a minimizar o erro da rede. Em uma próxima iteração, novamente são apresentadas as imagens da base de treinamento à rede, um novo valor de erro é calculado e os pesos são reajustados. Após subsequentes iterações, chamadas de épocas de treinamento, é esperado que o erro convirja para um valor menor (GOOGLE, 2020b). Este procedimento de treinamento da rede é chamado de *backpropagation* (tradução livre, retropropagação) e, embora não seja o único método de treinamento, é amplamente utilizado.

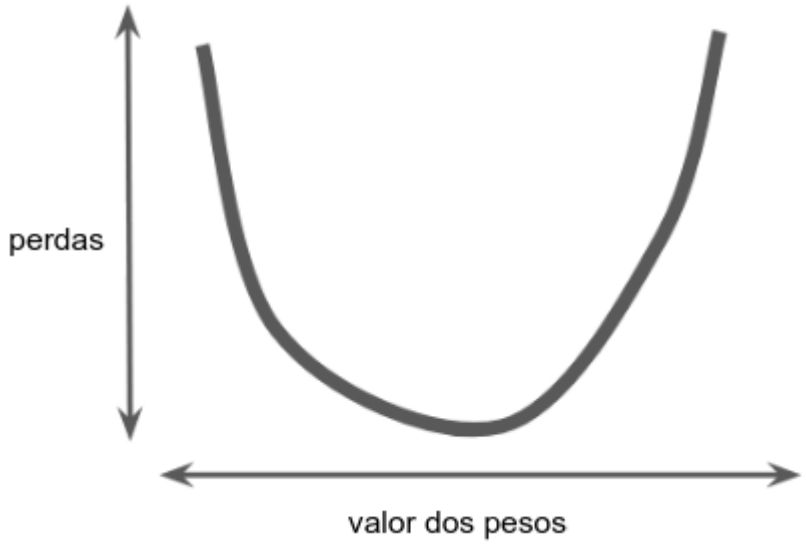
Nesse processo, duas decisões de arquitetura devem ser feitas, sendo elas a escolha da função para cálculo do erro e do método de mudança dos pesos.

A função para cálculo do erro é chamada de função de perda. Entropia cruzada é a função de perda mais usada atualmente em redes neurais para problemas de classificação (BROWNLEE, 2019).

Com a função de erro definida, deve-se tentar, a cada iteração, diminuí-lo. Assim, tem-se um problema de otimização. Observando o gráfico da Figura 10 que representa, de forma simplificada, a relação de erro da rede com os seus pesos, o objetivo é encontrar os valores dos pesos que minimizem o erro.

A solução mais amplamente usada é o gradiente descendente. Por este método, é calculado o gradiente do erro a partir da função de perda para indicar a direção em que os pesos devem ser ajustados (aumentar ou diminuir os seus valores). Os pesos são atualizados a cada iteração no sentido do gradiente em que o erro é reduzido (GOOGLE, 2020b).

Figura 10 – Curva de erro da rede

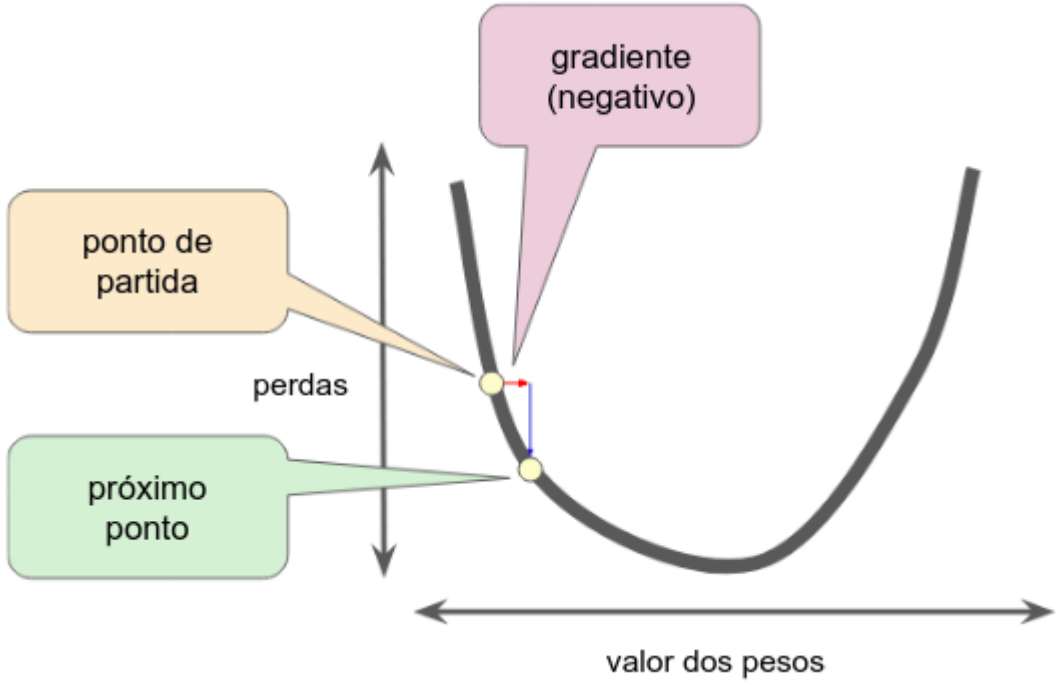


Fonte: Google (2020).

Nota: Modificado pela autora.

Para o entendimento do método gradiente descendente pode-se utilizar a Figura 11 como ilustração. Nesta Figura, a partir de um ponto inicial (configuração inicial dos pesos da rede), é calculado o gradiente do erro. O gradiente aponta na direção e sentido em que o valor do erro aumenta mais rápido. Portanto, como o objetivo é minimizar o erro, os pesos são atualizados no sentido oposto (negativo do gradiente). O tamanho dos passos para ajustar os pesos em cada iteração é definido pela taxa de aprendizado da rede. Um gradiente igual ou próximo a zero indica que a rede alcançou um mínimo local ou global da superfície de erro. Se a taxa de aprendizado for elevada, o treinamento da rede pode convergir rapidamente para uma solução não muito boa e oscilar mais o desempenho da rede durante o treinamento, por outro lado, se for muito baixo, ele pode encontrar uma solução melhor, mas demorar muito para convergir para uma solução.

Figura 11 – Demonstração gráfica do método gradiente descendente



Fonte: Google (2020).

Nota: Modificado pela autora.

2.3 *Transfer Learning*

Em geral, as redes neurais convencionais são treinadas do zero, ou seja, tendo início com pesos aleatórios, para resolver uma determinada tarefa, porém, nas CNN, devido a sua arquitetura com muitas camadas e a necessidade de uma quantidade grande de imagens para treinamento, o procedimento de treinamento do zero pode ser muito custoso e até mesmo inviável. Assim, para estas arquiteturas, é mais interessante utilizar alguma forma de reaproveitamento dos pesos obtidos em um treinamento anterior, de forma a poupar tempo e esforço computacional.

O reaproveitamento de aprendizado pode ser implementado nas redes profundas através do *transfer learning* (tradução, transferência de aprendizado) (SARKAR, 2018). Assim, uma rede que foi treinada para realizar uma tarefa aqui representada por **X** pode ter a sua arquitetura e os seus pesos reaproveitados por *transfer learning* para realizar uma outra tarefa **Y**. Quanto mais próxima for a tarefa **Y** de **X**, maior a probabilidade da rede apresentar um desempenho em **Y** semelhante ao obtido em **X**. Isso porque os conceitos menos complexos, como bordas, formas e características

simples, necessários para a rede realizar a tarefa em ambos os domínios (**X** e **Y**) são semelhantes (ROMAN, 2020). Quando as tarefas são de domínio semelhante, por exemplo, imagens naturais, mas com objetivos diferentes, como **X** sendo a identificação de corpos humanos e **Y** a identificação de faces, por exemplo, pode-se utilizar um procedimento chamado de *fine-tuning* (tradução, ajuste fino), cujo objetivo é fazer um ajuste fino nos pesos para aprimorar o desempenho da rede (ROMAN, 2020).

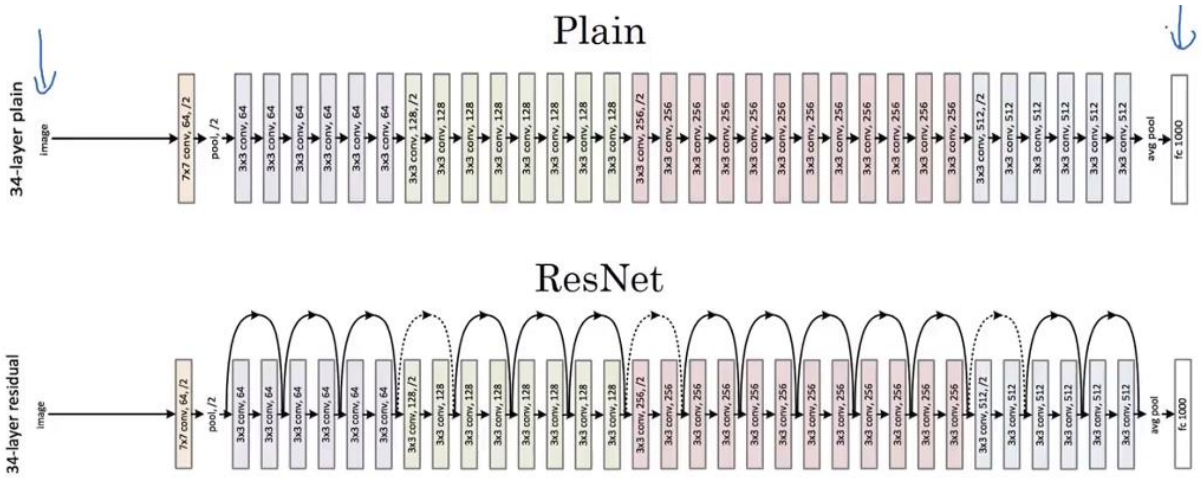
Na prática, o *transfer learning* é usado quando há uma pequena quantidade de dados (em comparação ao número de parâmetros da rede que precisam ser ajustados) para realizar a tarefa, mas existem à disposição redes treinadas com grandes quantidades de dados de natureza semelhante ao da tarefa alvo. Há diversas bibliotecas de *machine learning* e aprendizado profundo que disponibilizam redes previamente treinadas para o uso de *transfer learning*, como as redes Resnet, usadas neste trabalho e descritas na seção a seguir.

2.4 Resnet

Conforme a profundidade de uma CNN aumenta, é intuitivo pensar que características mais complexas conseguem ser identificadas em uma imagem, porém, foi observado que o desempenho de treinamento tende a piorar para redes convencionais quando possuem um número muito grande de camadas. Esse resultado decorre de problemas como a explosão e o desaparecimento do gradiente (DEEP LEARNING BOOK, 2019). Como solução, a arquitetura de rede residual (ResNet, do inglês *residual network*) foi proposta por Simonyan e Zisserman (SIMONYAN et al., 2014). Essa arquitetura tem como principal característica ter suas camadas, além de conectadas diretamente com as camadas adjacentes, conectadas com camadas mais profundas da rede através de *shortcut connections* (conexões de atalho) (HE; ZHANG; REN; SUN, 2016a). Isto pode ser observado na Figura 12, que expõe a arquitetura de uma ResNet 34, onde o *shortcut connection* é feito em blocos de 2 camadas de extração de características, ou seja, 2 camadas convolucionais seguidas de uma função de ativação (SHORTEN, 2019). Na Resnet 50, Resnet 101 e Resnet 152, além dos blocos de 2 camadas, existem blocos de 3 camadas entre

os *shortcuts connections* (HE; ZHANG; REN; SUN, 2016b). Por meio dessa técnica é possível o uso de redes convolucionais com um número muito maior de camadas.

Figura 12 – Arquitetura de uma Resnet comparada com uma CNN convencional



Fonte: He, Zhang, Ren e Sun (2016a).

As ResNet 50, ResNet 101 e ResNet 152 são, respectivamente, redes ResNet com 50, 101 e 152 camadas. Neste trabalho são usados modelos com essas três arquiteturas treinadas com a base de dados pública ImageNet. Esta base de dados foi criada para fins de pesquisa em 2009 e contém atualmente 14.197.122 imagens distribuídas em 21.841 classes (IMAGENET, 2020).

2.5 Pytorch

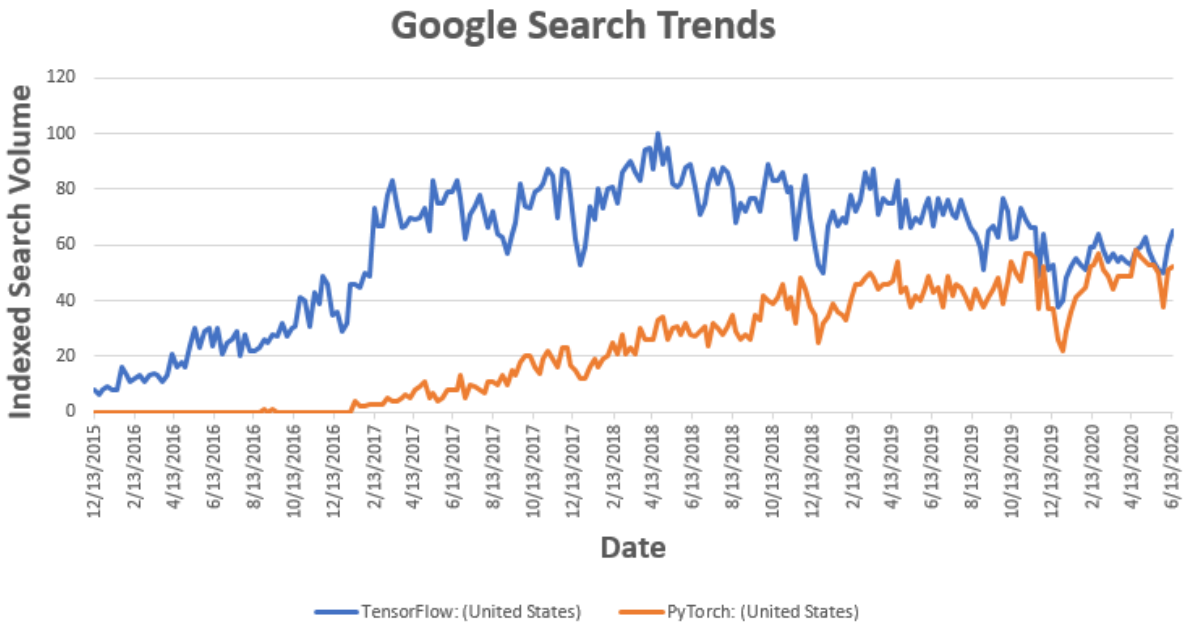
O Pytorch é uma biblioteca otimizada de *machine learning*. Seu desenvolvimento teve início no laboratório de pesquisas de inteligência artificial do Facebook e hoje conta com uma enorme comunidade de desenvolvedores *open source* (PYTORCH, 2019a).

Grandes projetos da indústria usam o Pytorch no seu desenvolvimento, por exemplo, pode ser citado o Testa Autopilot (PYTORCH, 2019b).

O Pytorch tem se popularizado bastante na comunidade de *machine learning* e hoje em dia disputa o título de biblioteca mais usada para *deep learning* com Tensorflow, que há muito tempo domina esse nicho. Na Figura 13 pode ser vista a tendência de

pesquisa no Google de tópicos relacionados a Tensorflow e Pytorch. Isso mostra o crescimento do interesse no Pytorch nos últimos anos.

Figura 13 – Tendência de pesquisas no Google por Tensorflow e Pytorch



Fonte: Saifee (2020).

De acordo com um estudo realizado em 2019 por Horace He (2019), Pythorch tem sido a biblioteca mais citada em artigos científicos das maiores conferências de visão computacional, processamento de linguagem natural e *machine learning*. Nessa análise, Horace He (2019) constatou que isso se dá pela sua simplicidade, pela qualidade de sua API e seu bom desempenho.

A simplicidade da biblioteca está ligada a trivialidade da integração com o ecossistema do Python, por exemplo, a integração com NumPy e a facilidade de se debugar o código. A qualidade da API do Pytorch se deve ao seu *design* mais direto, pois o Tensorflow teve sua API mudada muitas vezes ao longo dos anos, deixando-a sua arquitetura mais desorganizada (HORACE, 2019).

Acerca do desempenho, pode ser citada a compilação JIT (*Just In Time*) que, no final de 2018, foi implementada no lugar da interpretação padrão do Python. Esse tipo de compilação JIT, em questão de desempenho, é o meio termo entre a interpretação e a compilação (HORACE, 2019). Além disso, o Pytorch tem uma

ferramenta chamada paralelismo de dados que é capaz de distribuir o trabalho computacional em múltiplos núcleos de CPU e GPU (PYTORCH, 2021).

2.6 API Google Maps

Google Maps é um serviço online de mapeamento desenvolvido pelo Google e publicado inicialmente em 2005. Além do serviço online, o Google também disponibiliza a API Google Maps, também chamada de Google Maps Platform, que se divide em 17 serviços, sendo esses do tipo *software development kit* ou API. Os serviços disponíveis podem ser divididos em 3 categorias principais: *Maps* (tradução, mapas) que oferece os recursos de mapas e *street view*, *Routes* (tradução, rotas) que oferece os recursos de direções, matriz de distâncias e estradas, e *Places* (tradução, locais) que oferece os recursos de detalhes do local, local atual, encontrar local, autocompletar local, geocodificação, geolocalização e fuso horário (PLATAFORMA GOOGLE MAPS, 2021b). Neste trabalho, o recurso *Maps* foi utilizado, de forma que com a latitude e longitude fornecidos como parâmetros de entrada, é obtida uma imagem de satélite da região. A API Google Maps é um serviço pago e seu custo varia de acordo com o seu uso, sendo cobrado de forma mensal (PLATAFORMA GOOGLE MAPS, 2021c).

As imagens de satélite usadas no Google Maps são atualizadas com uma frequência que pode variar entre 1 a 3 anos. Essa variação acontece de forma que cidades que mudam mais rapidamente, ou seja, cidades maiores são atualizadas com uma frequência maior que cidades menores (GOOGLE, 2020a). As imagens usadas neste trabalho tiveram sua mais recente atualização em 2021. Isso permite que sistemas que utilizem tal ferramenta fiquem atualizados com uma certa periodicidade.

3 SOLUÇÃO PROPOSTA

3.1 Introdução

O projeto aqui apresentado visa a construção de uma CNN que seja capaz de classificar uma dada imagem de entrada, sendo esta uma imagem de satélite, como possuindo ZEIS ou não. Para o desenvolvimento foi necessário um estudo a fundo de técnicas de *deep learning*, como a construção de uma CNN, *transfer learning*, otimização, taxa de aprendizado, entre outros. Além disso, foi preciso construir a base de dados usada para o treino e teste da rede por meio de ferramentas disponíveis pela API Google Maps. Os dados coletados são escassos e desbalanceados, devido a isso, com o intuito de melhorar a qualidade e tamanho da base de dados criada, foi importante o uso de técnicas de aumento de dados nos dados de entrada usados para o treino do modelo.

3.2 Dados

3.2.1 Natureza dos Dados

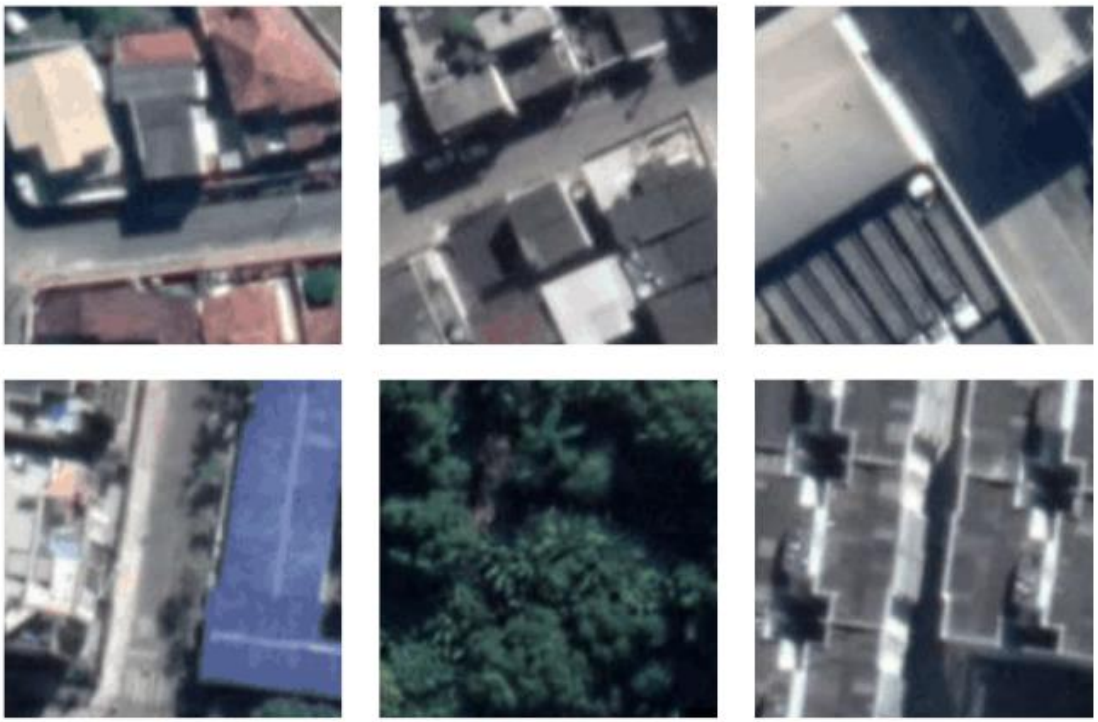
A base de dados, usada como entrada da rede, consiste em imagens de satélite coletadas em 2021 no formato PNG com dimensões de 155 x 155 *pixels*. As imagens foram extraídas da API do Google Maps das regiões urbanas de Vitória, Serra e Vila Velha no Estado do Espírito Santo. Nas Figuras 14 e 15 podem ser vistos alguns exemplos.

Figura 14 – Imagens de satélite contendo ZEIS (vazios urbanos)

Fonte: Plataforma Google Maps (2021a).

Nota: Modificada pela autora.

Figura 15 – Imagens de satélite que não contém ZEIS (vazios urbanos)



Fonte: Plataforma Google Maps (2021a).

Nota: Modificada pela autora.

3.2.2 Criação da base de dados

As imagens usadas neste projeto foram obtidas pela API do Google Maps através da biblioteca *Python Client for Google Maps Services* (GITHUB, 2020). Primeiro foram geradas posições geográficas aleatórias dentro das regiões urbanas das cidades estudadas. Com as latitudes e longitudes em mãos, usando a API, foram coletadas imagens de satélite com dimensões de 800 x 800 *pixels* das localidades, como as expostas na Figura 16.

Figura 16 – Imagens de satélite retornadas pela API do Google Maps

Fonte: Plataforma Google Maps (2021a).

As ZEIS presentes nas imagens foram marcadas de forma aproximada com base nos critérios definidos no artigo “Elaboração De Critérios Para Delimitação De ZEIS De Vazios Urbanos No Município De Vila Velha” (CONTARATO, 2019). Cada uma das imagens obtidas foi mapeada como mostra o exemplo da Figura 17 abaixo. Com o mapeamento, foi estudada uma alternativa de projeto de segmentação de imagens foi estudada, porém, tendo em vista a quantidade de imagens obtidas e o tempo disponível para elaboração do projeto, foi necessário fazer um mapeamento menos

preciso das imagens de satélite e seguir com a implementação de um projeto para a classificação.

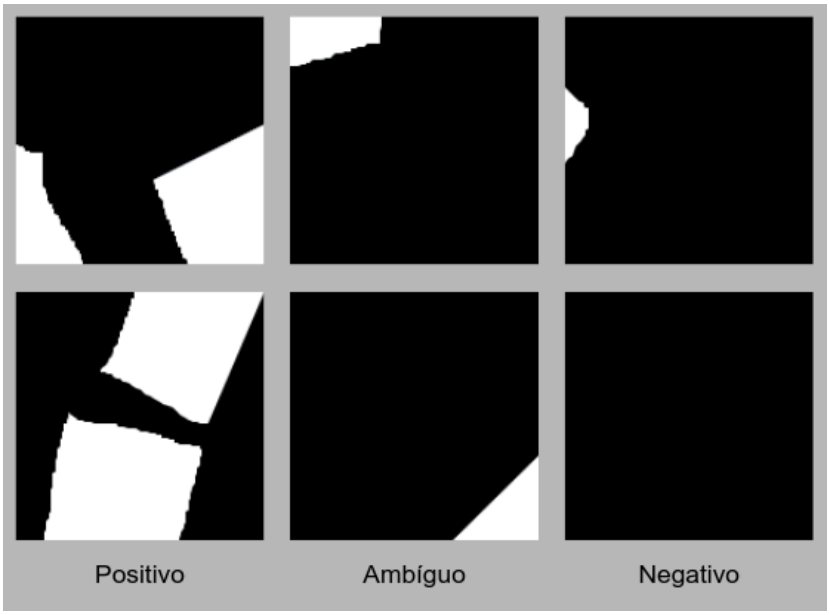
Figura 17 – Imagem de satélite e seu respectivo mapeamento

Fonte: Plataforma Google Maps (2021a).

Nota: Modificada pela autora.

Foram gerados 16 *patches* de dimensões 155 x 155 *pixels* de cada uma das imagens de satélite que foram classificados em 3 categorias de acordo com a imagem mapeada, como ilustra a Figura 18. Os *patches* com mais de 10% de sua área contendo uma ZEIS foram classificados como positivos, os *patches* com menos de 5% de sua área contendo uma ZEIS foram classificados como negativos e os *patches* contendo ZEIS entre 5% e 10% de sua área foram classificados como ambíguos e não foram incluídos na base de dados final.

Figura 18 – Classificação dos *patches*



Fonte: Produção da própria autora.

3.2.3 Pré-processamento

Com o intuito de melhorar a qualidade das imagens da base de dados, foi aplicada uma transformação de aumento de nitidez sobre cada imagem através do método de máscara de nitidez (CAMBRIDGE IN COLOUR, c2020). O resultado obtido pode ser visto na Figura 19.

Figura 19 – Imagem antes e depois do aumento de nitidez

Fonte: Plataforma Google Maps (2021a).

Nota: Modificada pela autora.

3.2.4 Aumento de Dados

Técnicas de aumento de dados são usadas como solução para dados escassos. Tais técnicas aumentam o tamanho e a qualidade da base de dados de entrada e, como resultado, são obtidos melhores modelos de *deep learning* (SHORTEN; KHOSHGOFTAAR, 2019).

No escopo de CNN, onde os dados de entrada são imagens, o aumento de dados consiste na aplicação de diferentes transformações nas imagens, podendo ser estas transformações simples, como espelhamento vertical ou horizontal, *zoom*, translações, entre outras, ou operações mais complexas, como aplicação de filtros diversos nas imagens (SHORTEN; KHOSHGOFTAAR, 2019). Na Figura 20 é ilustrado um exemplo da aplicação de aumento de dados em uma imagem.

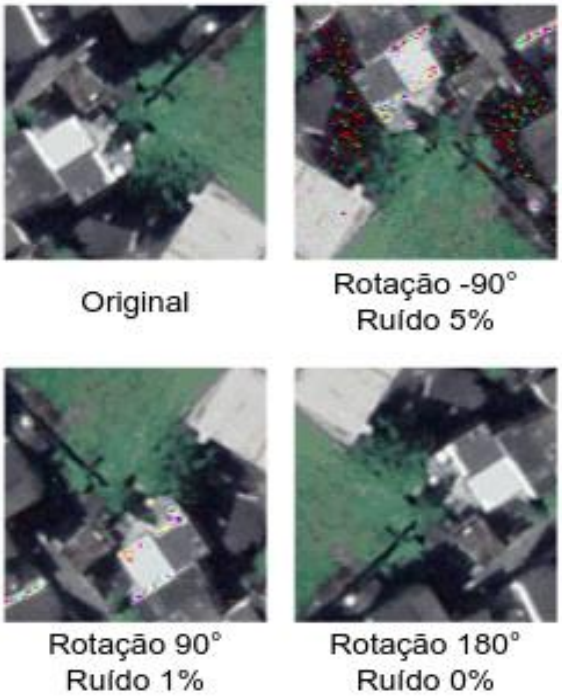
Figura 20 – Exemplo da técnica de aumento de dados

Fonte: Alto (2020).

Para este projeto foram aplicadas as transformações de rotação e adição de ruído nas imagens da classe mais escassa, isto é, para as imagens que contêm ZEIS somente no conjunto de dados usado para treino. Essa estratégia de sobreamostragem da classe minoritária tem como objetivo diminuir a disparidade das classes da base de dados ao mesmo tempo que aumenta o tamanho total da base de dados (BROWNLEE, 2020b). Dessa forma, cada imagem em que o aumento de dados é aplicado, dá origem a 4 imagens, sendo uma a imagem original

e as três restantes sua cópia rotacionada em 90, -90 e 180 graus e com adição de ruído com opacidade aleatoriamente escolhida entre 0%, 1% e 5%, como ilustra a Figura 21.

Figura 21 – Exemplo da técnica de aumento de dados usada no projeto



Fonte: Plataforma Google Maps (2021a).
Nota: Modificada pela autora.

Com a sobreamostragem da classe minoritária, o conjunto de dados usado para treino da rede passou a ser formado por 7249 imagens, sendo 39,3% classificadas como positivas e 67,6% como negativas em contraste com as 5038 imagens com 14,6% classificadas como positivas e 85,4% como negativas que formavam a base de dados antes da aplicação do aumento de dados.

3.3 Treino

Os dados foram divididos em amostras estratificadas de 70% para treino, 15% para validação e 15% para teste. No conjunto de treino foi aplicado a técnica de aumento de dados. A quantidade exata de dados usados para treino, teste e validação pode

ser observada no Quadro 1. O tamanho do mini *batch* definido para a rede foi de 32 imagens.

Quadro 1 – Quantidade de dados usados para treino, teste e validação

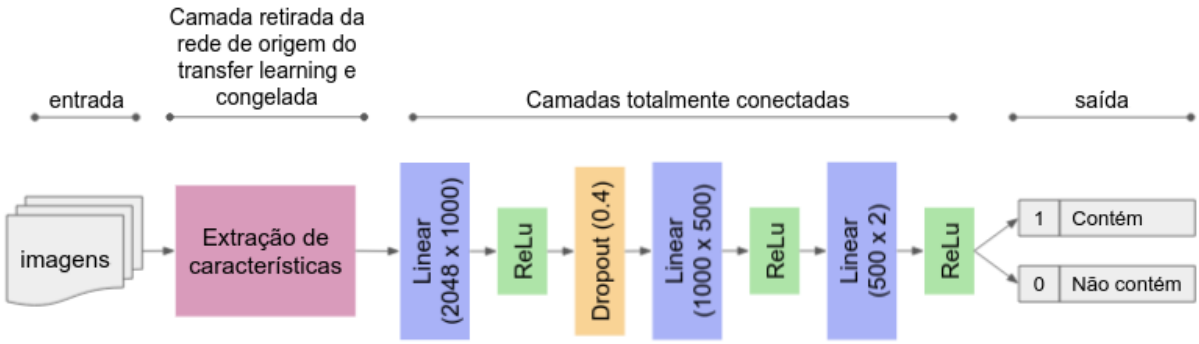
Divisão da base de dados	Quantidade de imagens	Classe positiva	Classe negativa
Treino	7249	40,7%	59,3%
Teste	1079	14,6%	85,4%
Validação	1079	14,6%	85,4%

Fonte: Produção da própria autora.

O número de épocas de treino é definido dinamicamente pela estratégia de *early stopping* (BROWNLEE, 2018) que conclui o treinamento de acordo com uma métrica monitorada da rede que nesse projeto foi definido como *F1-score*. O treino é finalizado quando a métrica monitorada permanece por um número definido de épocas, sendo este parâmetro chamado de paciência, estável ou com seu valor piorando (diminuindo ou aumentando dependendo da métrica). Isso indica que o treino convergiu ou que está divergindo novamente devido ao *overfitting* (sobreajuste) da rede. A paciência para o *early stopping* estipulado para este trabalho foi de 3 épocas.

A técnica de *transfer learning* é aplicada com as redes ResNet 50, ResNet 101 e ResNet 152 para comparação. No *transfer learning* as camadas convolucionais da rede são congeladas e a camada totalmente conectada é substituída pela sequência exposta no diagrama da Figura 22.

Figura 22 – Diagrama da rede usada no projeto



Fonte: Produção da própria autora.

A estratégia de gradiente descendente foi usada para a otimização da rede com entropia cruzada como a função de perda. A taxa de aprendizado é definida de forma dinâmica através da biblioteca *torch_lr_finder* que usa a estratégia criada por Leslie N. Smith (2017) e demonstrada no artigo *"Cyclical Learning Rates for Training Neural Networks"* para encontrar a taxa de aprendizado ótima da rede. E, por último, é definido o decaimento exponencial de 0,9 da taxa de aprendizado (LAU, 2017), de modo que a cada época a taxa de aprendizado será multiplicada por 0,9.

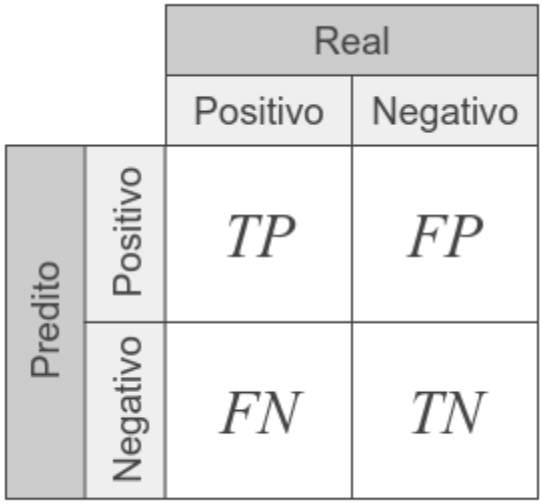
3.4 Teste

Para avaliação da qualidade do modelo obtido após cada treino, são usadas quatro métricas: precisão, *recall, F1-score* e acurácia. Para o cálculo de tais métricas, são empregados os valores encontrados na matriz de confusão do teste da rede (SUNASRA, 2017).

A matriz de confusão é uma tabela que, no caso de uma classificação binária, tem 2 colunas e 2 linhas e permite a visualização do desempenho de uma rede de classificação. As colunas da tabela representam a classificação real dos dados e as linhas representam a classificação feita pela rede. Dessa forma, a matriz de confusão, como mostra a Figura 23, possui quatro medidas: os verdadeiros positivos (TP, do inglês *true positive*) e verdadeiros negativos (TN, do inglês *true negative*), que tem a classe predita pelo modelo igual a classe real da imagem, sendo esse valor positivo e negativo respectivamente. Os falsos positivos (FP, do inglês *false positive*) e falsos negativos (FN, do inglês *false negative*), que tem prediz a classe

positiva e negativa respectivamente, porém de forma errônea, isto é, diferente da classe real da imagem em questão.

Figura 23 – Matriz de confusão



Fonte: Sunasra (2017).

Fonte: Adaptado pela autora.

A precisão (P) é a métrica que expõe a proporção na qual a predição da classe positiva é feita corretamente (MACHINE LEARNING, 2020). Essa medida também pode ser entendida como a confiabilidade do modelo (POWERS, 2007) em classificar uma imagem como um positivo (que contém ZEIS). A precisão é calculada pela Equação 2.

$$P = \frac{TP}{TP + FP} \times 100\% \quad (2)$$

Já o *recall* (R) é a métrica que descreve a sensibilidade do modelo (POWERS, 2007). Ela mede a proporção na qual a classe positiva foi identificada corretamente (MACHINE LEARNING, 2020). O *recall* expõe uma proporção de acertos relativa à classe positiva presente na base de dados em contraste com a precisão, que expõe uma proporção de acertos em relação ao resultado final predito pela rede. O *recall* é calculado conforme a Equação 3.

$$R = \frac{TP}{TP + FN} \times 100\% \quad (3)$$

O *F1-score* (F), como mostrado na Equação 4, é a métrica que leva em consideração a precisão e o *recall* por meio de uma média harmônica dos dois valores (SASAKI, 2007). Por esse motivo, o *F1-score* foi escolhido como principal métrica para avaliação do desempenho das redes treinadas neste trabalho.

$$F = \frac{P \times R \times 2}{P + R} \times 100\% \quad (4)$$

A métrica de avaliação de desempenho mais comumente usada é a acurácia (Acc). Essa métrica, como mostra a Equação 5, é a razão entre elementos corretamente classificados e o número total de elementos da base de dados. Para este trabalho, a acurácia não tem tanto valor significativo devido ao uso de uma base de dados desbalanceada. Neste caso, a acurácia não é recomendada para a avaliação dos modelos por apresentar um resultado enviesado (BROWNLEE, 2020a).

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \quad (5)$$

4 RESULTADOS

Neste capítulo são apresentados os resultados dos experimentos realizados no trabalho. O intuito dos experimentos é encontrar um modelo de CNN que obtenha desempenho satisfatório para resolver o problema de identificação de ZEIS em imagens de satélite.

Para todos os treinos realizados foram definidos como hiperparâmetros fixos o mini *batch* de 32 imagens, decaimento exponencial de 0,9 na taxa de aprendizagem a cada época e paciência do *early stopping* de 3 épocas.

Todas as redes usadas foram previamente treinadas com a base de dados pública ImageNet (IMAGENET, 2020) e, por meio de *transfer learning*, foram usadas para realizar a tarefa deste trabalho.

4.2 Aumento de Dados

Com o intuito de observar o impacto do aumento de dados na qualidade do modelo, a rede ResNet 50 foi treinada com a base de dados antes e após a aplicação da técnica. Em ambos os treinos todos os hiperparâmetros fixos foram mantidos iguais, porém, a taxa de aprendizado, por ser um hiperparâmetro calculado dinamicamente antes do treino, é diferente para o treino com e sem o aumento de dados. Foi aplicado *transfer learning* para a rede Resnet 50 onde foram congelados os pesos das camadas convolucionais e treinado somente as últimas camadas, conforme Figura 22. Esta medida garantiu redução do tempo de cada época de treino. No Quadro 2 são apresentados os valores definidos para o treinamento.

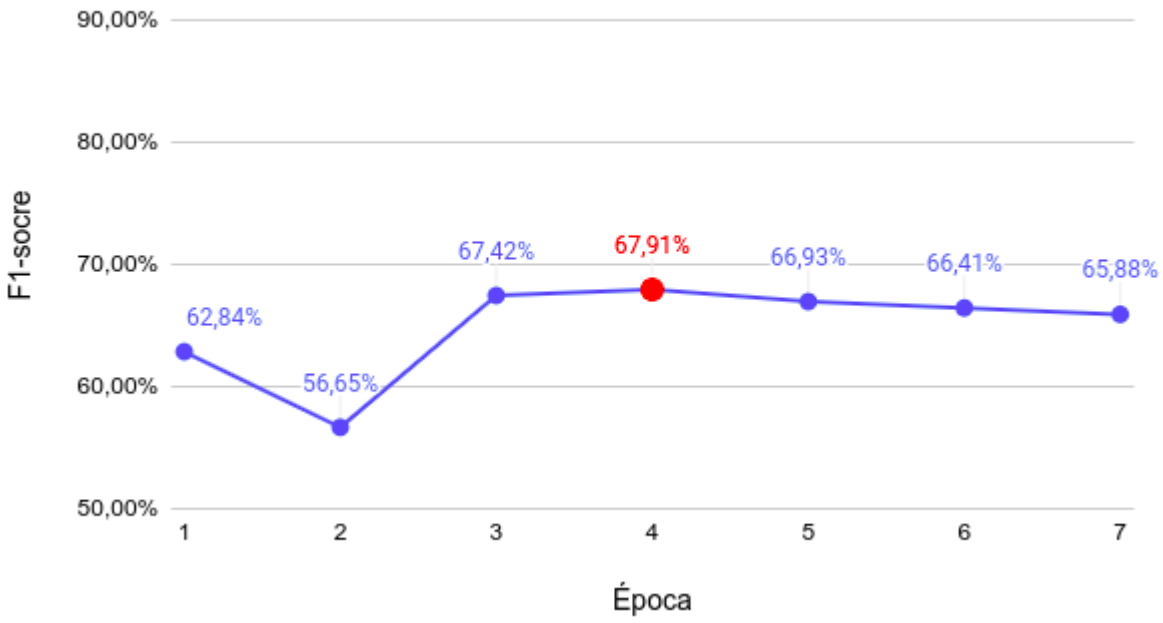
Quadro 2 – Valores dinâmicos dos treinos com e sem aumento de dados

Aumento de dados	Taxa de aprendizagem	Épocas de treino	Época de melhor resultado
Sim	7,05E-03	14	11
Não	5,34E-03	7	4

Fonte: Produção da própria autora.

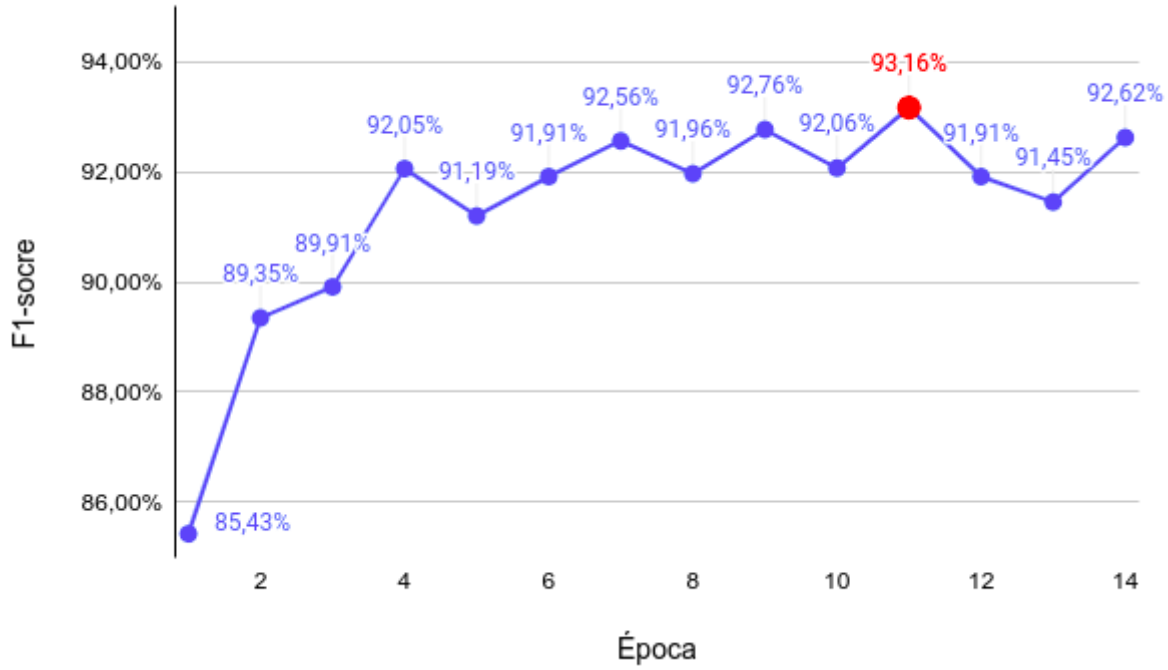
As Figuras 24 e 25 mostram o *F1-score* da rede após cada época de treino. Para a rede sem aumento de dados, o *F1-score* converge na época 4 e começa a divergir nas posteriores, quando o mecanismo de *early stopping* encerra o treino. O mesmo acontece para a rede com aumento de dados na época 11. A rede treinada sem aumento de dados apresentou *F1-score* significativamente inferiores que a rede treinada com aumento de dados.

Figura 24 – F1-score por época no treino sem uso de aumento de dados



Fonte: Produção da própria autora.

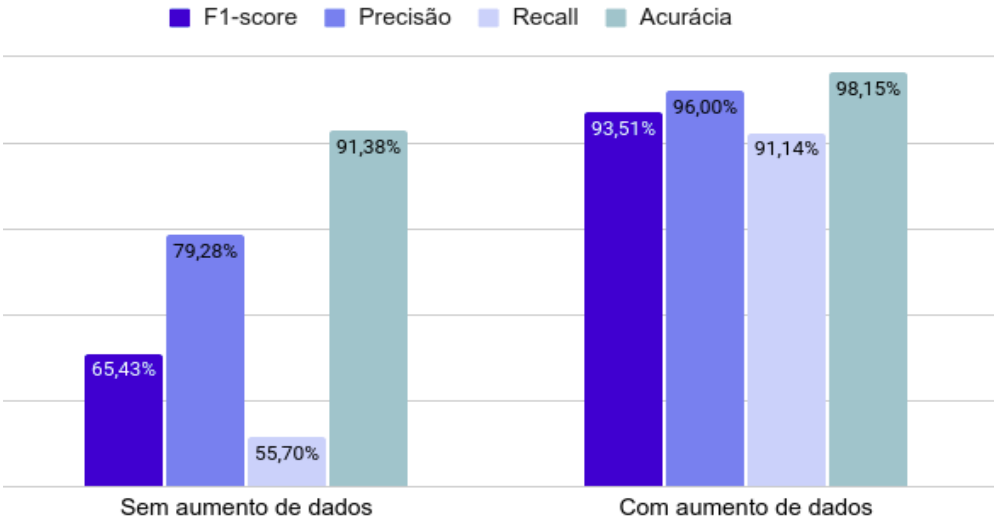
Figura 25 – F1-score por época de treino com uso de aumento de dado



Fonte: Produção da própria autora.

Com os modelos gerados na época em que cada treino convergiu, foram calculadas a precisão, *recall*, acurácia e *F1-score* sobre o conjunto de dados de teste. Tais métricas são exibidas na Figura 26. A rede treinada com a base de dados após o aumento de dados obteve melhores resultados em todas as métricas calculadas com uma diferença expressiva nos resultados, quando comparada com a rede treinada sem aumento de dados. Isto evidencia a importância da técnica de aumento de dados para a realização da tarefa.

Figura 26 – Comparação das métricas de avaliação das redes treinadas com e sem aumento de dados



Fonte: Produção da própria autora.

4.3 Transfer *Learning*

Para definir a melhor rede para se usar como origem na técnica de *transfer learning*, foram testadas as redes Resnet 50, Resnet 101 e Resnet 152. Novamente, os hiperparâmetros foram mantidos idênticos com exceção à taxa de aprendizagem e o número de épocas, sendo estas definidas de forma dinâmica. A base de dados para os treinos, em que foi aplicado o aumento de dados, e para os testes também foram mantidos os mesmos. Abaixo no Quadro 3 estão expostos os valores dinâmicos determinados pelo código para o treinamento de cada rede, assim como a época na qual o melhor resultado da rede foi alcançado.

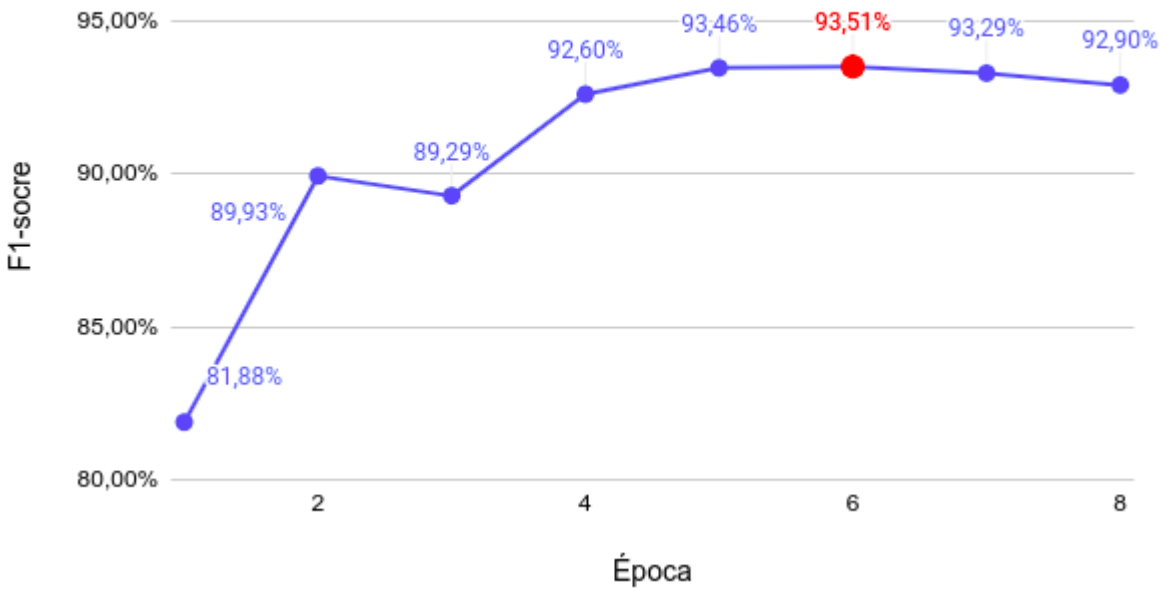
Quadro 3 – Valores dinâmicos dos treinos com diferentes redes com *transfer learning*

Rede	Taxa de aprendizagem	Épocas de treino	Época de melhor resultado
Resnet 50	7,05E-03	14	11
Resnet 101	6,14E-03	8	6
Resnet 152	6,14E-03	11	8

Fonte: Produção da própria autora.

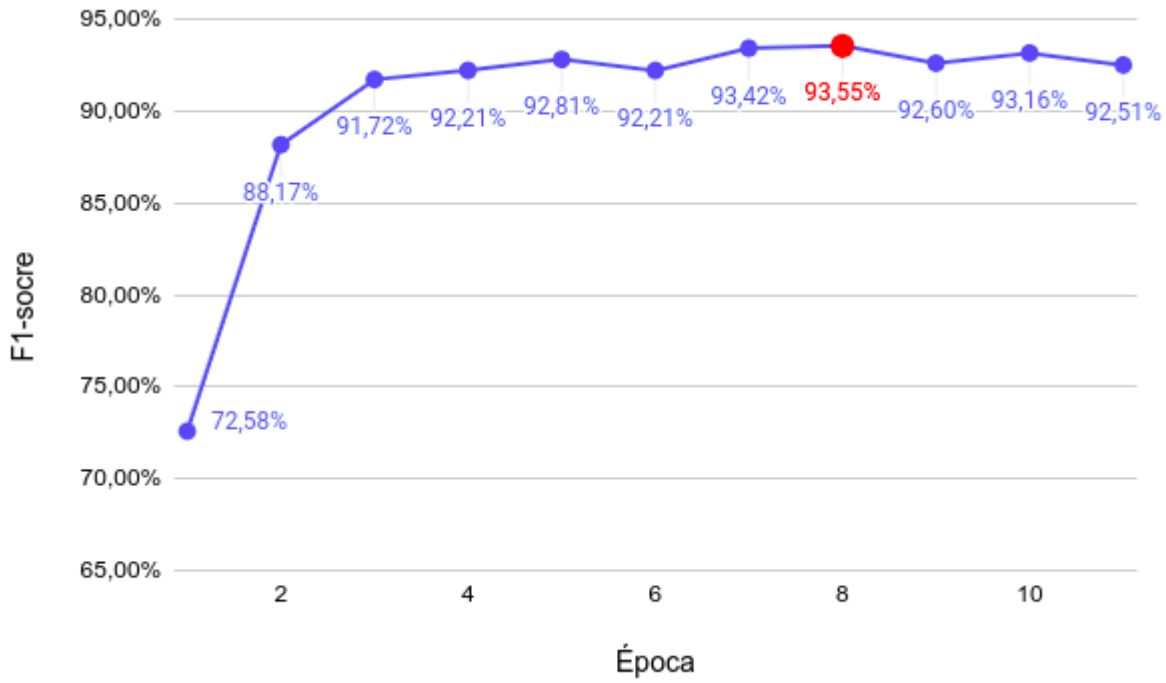
Nas Figuras 25, 27 e 28 são exibidos o *F1-score* por época do treinamento com *transfer learning* das redes Resnet 50, Resnet 101 e Resnet 152, respectivamente. O maior *F1-score* alcançado pelas redes não tem uma diferença muito expressiva, sendo o melhor resultado o da Resnet 152 com *F1-score* de 93,55% em contraste com 93,51% da Resnet 101 e 93,16% da Resnet 50.

Figura 27 – *F1-score* por época de treino com *transfer learning* da rede Resnet 101



Fonte: Produção da própria autora.

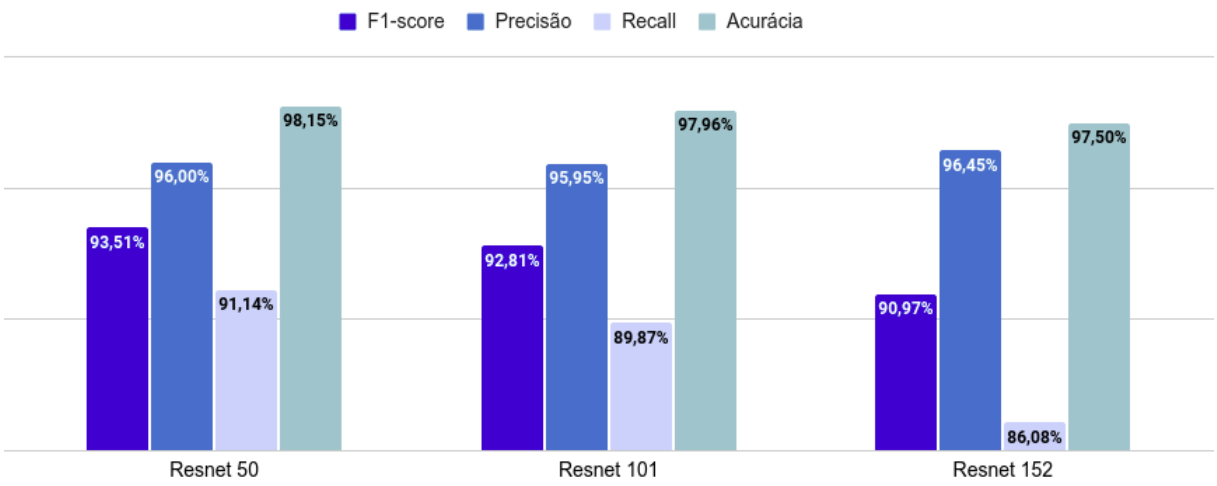
Figura 28 – *F1-score* por época de treino com *transfer learning* da rede Resnet 152



Fonte: Produção da própria autora.

Os valores de precisão, *recall*, acurácia e *F1-score* das redes com *transfer learning* utilizando as Resnet 50, Resnet 101 e Resnet 152 são mostrados na Figura 29. Apesar do *F1-score*, que é a principal métrica usada neste estudo, ser maior na validação com a Resnet 152, na base de dados usada para teste, o modelo que usou a Resnet 50 teve o melhor desempenho com um *F1-score* de 93,51% enquanto os modelos com a Resnet 101 e a Resnet 152 obtiveram um F1-score de 92,81% e 90,97% respectivamente.

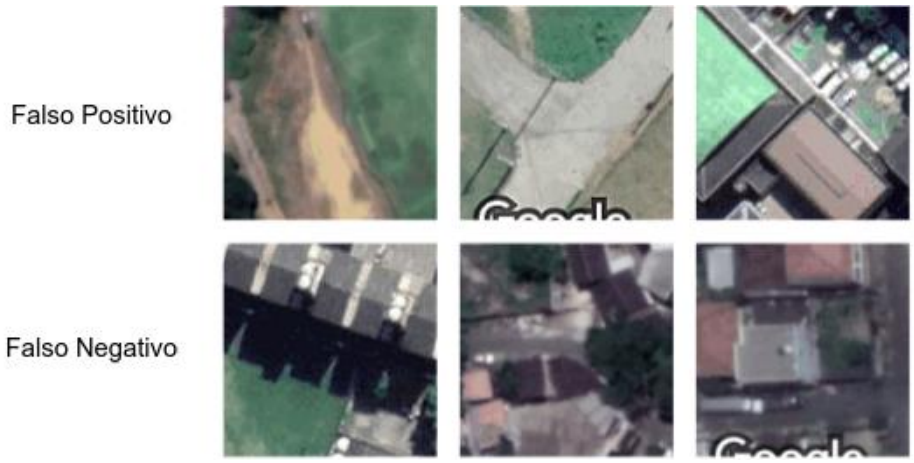
Figura 29 – Comparação das métricas de avaliação dos modelos usando *transfer learning* com diferentes redes



Fonte: Produção da própria autora.

Na Figura 30 são exibidas algumas imagens que foram erroneamente classificadas pelo modelo com melhores métricas, isto é, o modelo que usou a Resnet 50. Pode-se destacar, no caso das imagens classificadas equivocadamente como negativas (FN), que sombras de edifícios sobre as ZEIS podem ser um dificultador para a classificação correta da imagem. Por outro lado, algumas das imagens falsamente rotuladas como positivas (FP) foram aquelas com campos de futebol no meio de áreas urbanas, o que é um ponto desfavorável para correta classificação. Aparentemente a rede associou que áreas verdes em meio à malha urbana é um dos indícios de ZEIS.

Figura 30 – Imagens erroneamente classificadas pelo modelo



Fonte: Plataforma Google Maps (2021a).

Nota: Modificada pela autora.

5 CONCLUSÃO

O problema de famílias situadas em zonas impróprias para moradia é uma questão muito presente em diversas cidades brasileiras e que afeta a qualidade de vida da população que se encontra nessa situação. Uma possível solução para esta questão se dá pelo aproveitamento de ZEIS a partir de programas governamentais. O primeiro passo para colocar tais planejamentos em prática é a catalogação das localidades das ZEIS nas cidades. Essa localização deve ser feita com certa frequência visto que as cidades estão em constantes mudanças. Uma barreira encontrada para essa etapa é o alto custo para a localização das ZEIS que atualmente são encontradas a partir de pesquisas de campo tradicionais.

O uso do processamento de imagens de satélite e aprendizado de máquina para auxiliar neste processo demandaria custos menores e, por consequência, tornaria possível uma maior frequência na atualização dos dados. O objetivo deste trabalho foi propor uma rede CNN para localização de ZEIS na Grande Vitória. Os objetivos propostos foram alcançados, e a melhor rede usada alcançou uma precisão de 91,14% e *recall*, ou sensibilidade, de 96,00%. Foi verificado que o desempenho da rede teve relevante melhoria com a implementação de técnicas de aumento de dados. O estudo feito também revelou que o *transfer learning* com a rede Resnet 50 se mostrou superior ao uso das redes Resnet 101 e Resnet 152 para a base de dados criada. Com a identificação de possíveis regiões de ZEIS nas imagens é possível saber, pela geolocalização, a localização aproximada da região e especialistas podem conferir o local.

Para trabalhos futuros, o uso de *fine-tuning* com a Resnet pode ser aplicado para aumentar o desempenho do modelo. Além disso, a tarefa de classificação de imagens pode ser substituída por uma de segmentação de imagens usando redes neurais, o que permite uma localização mais precisa das ZEIS nas imagens. Para isto, é necessária uma marcação mais precisa nas imagens.

É interessante também a construção de um banco de imagens com base em imagens de satélite e infravermelho fornecidas pelo Instituto Nacional de Pesquisas Espaciais (INPE) para o treino da rede, assim como a técnica de aumento de dados

pode ser mais elaborada com outros tipos de transformações nas imagens como, por exemplo, o espelhamento da imagem na horizontal e vertical e filtros de cor.

Outro projeto futuro possível, focado mais no uso prático do modelo, é a implementação de uma plataforma *web* que possibilite a classificação de uma área como contendo ou não uma ZEIS através da integração com a API Google Maps e o modelo desenvolvido neste trabalho. Dessa forma, um usuário poderia, através de uma coordenada ou conjunto de coordenadas geográficas fornecidas, obter quais localidades contêm possíveis ZEIS de acordo com a classificação do modelo.

REFERÊNCIAS

ABE, M. C. **Minha Casa Minha Vida Dez Anos**. UOL. 2019. Disponível em: https://economia.uol.com.br/reportagens-especiais/minha-casa-minha-vida-dez-anos. Acesso em: 6 set. 2021.

ALTO, V. **Data Augmentation in Deep Learning**. 2020. Disponível em: https://medium.com/analytics-vidhya/data-augmentation-in-deep-learning-3d7a539f7a28. Acesso em: 6 set. 2021.

BRASIL. **Lei nº. 10.257, de 10 de julho de 2001**.Casa Civil, 2001. Disponível em: http://www.planalto.gov.br/ccivil_03/leis/leis_2001/l10257.htm. Acesso em: 6 set. 2021.

BROWNLEE, J. **A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks**. 2018. Disponível em: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/. Acesso em: 6 set. 2021.

BROWNLEE, J. **Failure of Classification Accuracy for Imbalanced Class Distributions**. 2020a. Disponível em: https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/. Acesso em: 6 set. 2021.

BROWNLEE, J. **How to Choose Loss Functions When Training Deep Learning Neural Networks**. 2019. Disponível em: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/. Acesso em: 6 set. 2021.

BROWNLEE, J. **Random Oversampling and Undersampling for Imbalanced Classification**. 2020b. Disponível em: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/. Acesso em: 6 set. 2021.

CONTARATO, B. **Elaboração de Critérios para Delimitação de ZEIS de Vazios Urbanos no Município de Vila Velha.** 2019. Disponível em: https://issuu.com/brenda.contarato/docs/tcc_ii_brenda_contarato_publica__o. Acesso em: 6 set. 2021.

CAMBRIDGE IN COLOUR. **Guide to Image Sharpening**. c2020. Disponível em: https://www.cambridgeincolour.com/tutorials/image-sharpening.htm. Acesso em: 6 set. 2021.

COSTA, S.; CHAVES, F. **O Dever do Estado em Promover a Ocupação Urbana e a Garantia à Moradia no Estatuto da Cidade.** 2019. Disponível em:

https://jus.com.br/artigos/73582/o-dever-do-estado-em-promover-a-ocupacao-urbana-e-a-garantia-a-moradia-no-estatuto-da-cidade. Acesso em: 6 set. 2021.

DENG, L.; YU, D. **Deep Learning: Methods and Applications**. 2014. In: FOUNDATIONS AND TRENDS IN SIGNAL PROCESSING, 2014, v. 7, n. 3-4, p. 198-200.

DEEP LEARNING BOOK. **Outros Problemas com o Gradiente em Redes Neurais Artificiais**. 2019, Disponível em: https://www.deeplearningbook.com.br/outros-problemas-com-o-gradiente-em-redes-neurais-artificiais/. Acesso em: 6 set. 2021.

DERTAT, A. **Applied Deep Learning - Part 1: Artificial Neural Networks**. Towards Data Science. 2017a. Disponível em: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6. Acesso em: 6 set. 2021.

DERTAT, A. **Applied Deep Learning - Part 4: Convolutional Neural Networks**. Towards Data Science. 2017b. Disponível em: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2. Acesso em: 6 set. 2021.

ESPÍRITO SANTO. LEI COMPLEMENTAR N. 872, DE 07 DE DEZEMBRO DE 2017. **Institui o Plano de Desenvolvimento Urbano Integrado**. Disponível em: http://www3.al.es.gov.br/Arquivo/Documents/legislacao/html/lec8722017.html. Acesso em: 6 set. 2021.

FRAGMAQ. **Entenda o Conceito de Cidades Compactas e Como Elas Podem Solucionar Problemas de Transporte.** 2019. Disponível em: https://www.fragmaq.com.br/blog/entenda-conceito-cidades-compactas-podem-solucionar-problemas-transporte/. Acesso em: 6 set. 2021.

FUNDAÇÃO JOÃO PINHEIRO. **Déficit Habitacional no Brasil 2016-2019.** 2021. Disponível em: http://fjp.mg.gov.br/wp-content/uploads/2021/04/21.05_Relatorio-Deficit-Habitacional-no-Brasil-2016-2019-v2.0.pdf. Acesso em: 6 set. 2021.

GHOLAMALINEZHAD, H.; KHOSRAVI, H. **Pooling Methods in Deep Neural Networks, a Review**. 2020. Disponível em: https://arxiv.org/pdf/2009.07485.pdf. Acesso em: 14 set, 2021.

GITHUB. **Python Client for Google Maps Services**. 2020. Disponível em: https://github.com/googlemaps/google-maps-services-python/blob/master/README.md. Acesso em: 14 set. 2021.

GOMES, L. **Machine-Learning Maestro Michael Jordan on the Delusions of Big Data and Other Huge Engineering Efforts**. 2014. Disponível em: https://spectrum.ieee.org/robotics/artificial-intelligence/machinelearning-maestro-michael-jordan-on-the-delusions-of-big-data-and-other-huge-engineering-efforts. Acesso em: 6 set. 2021.

GOOGLE. **Ask a Techspert: How do satellite images work?**. 2020a. Disponível em: https://blog.google/products/maps/how-do-satellite-images-work/. Acesso em: 14 set. 2021.

GOOGLE. **Reducing Loss: Gradient Descent**. 2020b. Disponível em: https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent. Acesso em: 6 set. 2021.

HE, K.; ZHANG, X.; REN, S.; SUN, J. **Deep Residual Learning for Image Recognition**. 2016a. Disponível em: https://arxiv.org/abs/1512.03385. Acesso em: 6 set. 2021.

HE, K.; ZHANG, X.; REN, S.; SUN, J. **Identity Mappings in Deep Residual Networks**. 2016b. Disponível em: https://arxiv.org/abs/1603.05027. Acesso em: 6 set. 2021.

HORACE, H. **The State of Machine Learning Frameworks in 2019**. The Gradient. 2019. Disponível em: https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/. Acesso em: 6 set. 2021.

IMAGENET. **About ImageNet**. 2020. Disponível em: https://image-net.org/about. Acesso em: 6 set. 2021.

INSTITUTO JONES DOS SANTOS NEVES. **Quem Somos**. 2019. Disponível em: http://www.ijsn.es.gov.br/institucional/quem-somos. Acesso em: 6 set. 2021.

JAIN, V. **Everything you need to know about “Activation Functions” in Deep learning models**. Towards Data Science. 2019. Disponível em: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253. Acesso em: 6 set. 2021.
KUNDUR, D. **Overlap-Save and Overlap-Add**. 2015. Disponível em: https://www.comm.utoronto.ca/~dkundur/course_info/real-time-DSP/notes/8_Kundur_Overlap_Save_Add.pdf. Acesso em: 6 set. 2021.

LAU, S. **Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning**. Towards Data Science. 2017. Disponível em:

https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1. Acesso em: 6 set. 2021.

LEE, H; GROSSE, R; RANGANATH, R; NG, A. Y. **Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations**. 2009. Disponível em: https://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf.Acesso em: 12 set. 2021.

MACHINE LEARNING. **Classification: Precision and Recall**. 2020. Disponível em: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall. Acesso em: 6 set. 2021.

NAVAMANI, T.M. **Deep Learning and Parallel Computing Environment for Bioengineering Systems**. 2019. Disponível em: https://www.sciencedirect.com/book/9780128167182/deep-learning-and-parallel-computing-environment-for-bioengineering-systems. Acesso em: 6 set. 2021.

PLATAFORMA GOOGLE MAPS. **API Google Maps**. 2021a. Disponível em: https://developers.google.com/maps/documentation/javascript/overview. Acesso em: 15 set. 2021.

PLATAFORMA GOOGLE MAPS. **Documentação**. 2021b. Disponível em: https://developers.google.com/maps. Acesso em: 6 set. 2021.

PLATAFORMA GOOGLE MAPS. **Maps JavaScript API Usage and Billing**. 2021c. Disponível em: https://developers.google.com/maps/documentation/javascript/usage-and-billing. Acesso em: 6 set. 2021.

POWERS, D. **Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation**. 2007. Disponível em: https://web.archive.org/web/20191114213255/https://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf. Acesso em: 6 set. 2021.

PREFEITURA MUNICIPAL DE VITÓRIA. **Plano de Metas 2018-2020**. 2017. Disponível em: https://www.vitoria.es.gov.br/arquivos/20180521_plano_de_metas.pdf. Acesso em: 6 set. 2021.

PREFEITURA MUNICIPAL DE VITÓRIA. **Plano de Metas 2021-2024**. 2020. Disponível em: http://planodemetas.vitoria.es.gov.br/. Acesso em: 6 set. 2021.

PYTORCH. **Pytorch Documentation**. 2019a. Disponível em: https://pytorch.org/docs/stable/index.html. Acesso em: 6 set. 2021.

PYTORCH. **PyTorch at Tesla - Andrej Karpathy, Tesla**. Youtube, 6 nov. 2019b. Disponível em: https://www.youtube.com/watch?v=oBklltKXtDE. Acesso em: 14 set. 2021.

PYTORCH. **Optional: Data Parallelism**. c2021. Disponível em: https://pytorch.org/tutorials/beginner/blitz/data_parallel_tutorial.html. Acesso em: 6 set. 2021.

ROHRER, B. **How convolutional neural networks work, in depth**. 2018. Disponível em: https://docs.google.com/presentation/d/1R-DnrghbU36jO8X4scbrrlx6gFyJHgSL3bD274sutng/edit#slide=id.p1. Acesso em: 6 set. 2021.

ROMAN, V. **CNN Transfer Learning & Fine Tuning**. Towards Data Science. 2020. Disponível em: https://towardsdatascience.com/cnn-transfer-learning-fine-tuning-9f3e7c5806b2. Acesso em: 6 set. 2021.

SAIFEE, M. **Pytorch vs Tensorflow in 2020**. Towards Data Science. 2020. Disponível em: https://towardsdatascience.com/pytorch-vs-tensorflow-in-2020-fe237862fae1. Acesso em: 6 set. 2021.

SARKAR, D. **A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning**. Towards Data Science. 2018. Disponível em: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a. Acesso em: 6 set. 2021.

SASAKI, Y. **The truth of the F-measure**. 2007. Disponível em: https://www.toyota-ti.ac.jp/Lab/Denshi/COIN/people/yutaka.sasaki/F-measure-YS-26Oct07.pdf. Acesso em: 6 set. 2021.

SHARMA, S. **Activation Functions in Neural Networks**. Towards Data Science. 2017. Disponível em: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Acesso em: 6 set. 2021.

SIMONYAN, K; ZISSERMAN, A. **Very Deep Convolutional Networks for Large-Scale Image Recognition**. 2014. Disponível em: https://arxiv.org/pdf/1409.1556.pdf. Acesso em: 1 out. 2021.

SHORTEN, C; KHOSHGOFTAAR, T. **A survey on Image Data Augmentation for Deep Learning**. 2019. Disponível em:

https://journalofbigdata.springeropen.com/track/pdf/10.1186/s40537-019-0197-0.pdf.Acesso em: 6 set. 2021.

SHORTEN, C. **Introduction to ResNets**. Towards Data Science. 2019. Disponível em: https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4. Acesso em: 12 set. 2021.

SMITH, L. Cyclical Learning Rates for Training Neural Networks. In: IEEE WINTER CONFERENCE ON APPLICATIONS OF COMPUTER VISION, 2017, Santa Rosa. **Proceedings** […]. Santa Rosa: IEEE, 2017. p. 464-472. Disponível em: https://ieeexplore.ieee.org/document/7926641. Acesso em: 6 set. 2021.

SUNASRA, M. **Performance Metrics for Classification problems in Machine Learning**. 2017. Disponível em: https://medium.com/@MohammedS/performance-metrics-for-classification-problemsin-machine-learning-part-i-b085d432082b. Acesso em: 6 set. 2021.

TECHWASTI. **CNN (Convolutional Neural Network) Using Gluon**. 2019. Disponível em: https://www.techwasti.com/cnn-convolutional-neural-network-using-gluon-88f4f7cccdb8/. Acesso em: 6 set. 2021.

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO. **Governo do Estado Lança Programas e ações de fomento à pesquisa e à inovação**. 2019. Disponível em: http://portal.ufes.br/conteudo/governo-do-estado-lanca-programas-e-acoes-de-fomento-pesquisa-e-inovacao. Acesso em: 6 set. 2021.

APÊNDICE A – CÓDIGOS

Quadro A1 - Arquivo utils.py

```
import numpy as np
import pandas as pd
import cv2 as cv
import os
from PIL import Image
import torchvision.transforms.functional as TF
import torch
import constants

def unsharp_mask(image, kernel_size=(5, 5), sigma=1.0, amount=1.0, threshold=0):
  blurred = cv.GaussianBlur(image, kernel_size, sigma)
  sharpened = float(amount + 1) * image - float(amount) * blurred
  sharpened = np.maximum(sharpened, np.zeros(sharpened.shape))
  sharpened = np.minimum(sharpened, 255 * np.ones(sharpened.shape))
  sharpened = sharpened.round().astype(np.uint8)

  if threshold > 0:
    low_contrast_mask = np.absolute(image - blurred) < threshold
    np.copyto(sharpened, image, where=low_contrast_mask)

  return sharpened


def get_f1_score(model, classFile, datasetDir):
  results = {
    "tp": [],
    "tn": [],
    "fp": [],
    "fn": [],
  }

  df = pd.read_csv(classFile)

  for index, row in df.iterrows():
    img = Image.open(datasetDir+"/"+row["IMAGE\LABEL"])
    imgt = TF.to_tensor(img)
    imgt = torch.unsqueeze(imgt, 0)

    out = model(imgt)

    if torch.argmax(out).data.numpy().tolist() == row["zeis"]:
      if row["zeis"] == 1:
        results["tp"].append(row["IMAGE\LABEL"])
      else:
        results["tn"].append(row["IMAGE\LABEL"])
    else:
```

```
    if row["zeis"] == 1:
      results["fn"].append(row["IMAGE\LABEL"])
    else:
      results["fp"].append(row["IMAGE\LABEL"])

  tp = len(results["tp"])
  tn = len(results["tn"])
  fp = len(results["fp"])
  fn = len(results["fn"])

  precision = tp / (tp + fp)
  recall = tp / (tp + fn)
  f1 = (2*precision*recall)/(precision+recall)
  return f1
```

Fonte: Produção da própria autora.

Quadro A2 - Arquivo validatemodel.py

```
import torch
import json
import pandas as pd
from PIL import Image
import torchvision.transforms.functional as TF
from myNetwork import model
import constants

# Carrega o modelo para teste
model.load_state_dict(torch.load("versions/26-50/model_10.pth"))

# Teste
model.eval()

df = pd.read_csv(constants.TEST_CLASS_FILE)

results = { "tp": [], "tn": [], "fp": [], "fn": [] }

for index, row in df.iterrows():
  img = Image.open(constants.TEST_DATASET+"/"+row["IMAGE\LABEL"])
  imgt = TF.to_tensor(img)
  imgt = torch.unsqueeze(imgt, 0)

  out = model(imgt)

  if torch.argmax(out).data.numpy().tolist() == row["zeis"]:
    if row["zeis"] == 1:
      results["tp"].append(row["IMAGE\LABEL"])
    else:
      results["tn"].append(row["IMAGE\LABEL"])
  else:
    if row["zeis"] == 1:
      results["fn"].append(row["IMAGE\LABEL"])
```

```
    else:
      results["fp"].append(row["IMAGE\LABEL"])

# Salva resultado
with open("artifacts/test_results.txt", "w") as file:
  file.write(json.dumps(results))
```

Fonte: Produção da própria autora.

Quadro A3 - Arquivo main do projeto

```
import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.optim as optim
import numpy as np

import constants
from utils import seconds_to_time_format, get_f1_score
from myNetwork import model
from myDataset import MyDataset
from EarlyStop import EarlyStopping

def train_algo(model):
  optimizer.zero_grad()
  output = model(image)
  loss = criterion(output, label)
  loss.backward()
  optimizer.step()

  return loss.item()

early_stopping = EarlyStopping(patience=3, min_delta=0.001)

# Dados de treino
TrainData = MyDataset(constants.TRAIN_DATASET, constants.TRAIN_CLASS_FILE)
Trainloader = DataLoader(TrainData,batch_size=constants.BACH_SIZE,shuffle=True)

# Otimizador
optimizer = optim.SGD(model.parameters(), lr=constants.LR)

# Função de perdas
criterion = nn.CrossEntropyLoss()

# Decaimento da taxa de aprendizado
decayRate = 0.9
my_lr_scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer=optimizer,
gamma=decayRate)

# Treino
```

```
for epoch in range(0, constants.EPOCHS):
  model.train()
  loss_train = 0

  # Loop de treino
  for idx, (image, label) in enumerate(Trainloader):
    loss = train_algo(model)
    loss_train+=loss

  # Validação
  model.eval()
  f1 = get_f1_score(model, constants.VALIDATION_CLASS_FILE,
constants.VALIDATION_DATASET)

  torch.save(model.state_dict(), "artifacts/model_{}.pth".format(epoch))

  early_stopping(1-f1)
  if early_stopping.early_stop:
    break

  my_lr_scheduler.step()
```

Fonte: Produção da própria autora.

Quadro A4 - Arquivo optimalLR.py

```
from torch.utils.data import DataLoader
import torch
import torch.nn as nn
import torch.optim as optim
from torch_lr_finder import LRFinder

import constants
from myNetwork import model
from myDataset import MyDataset

MainData = MyDataset(constants.TRAIN_DATASET, constants.TRAIN_CLASS_FILE)
Myloader = DataLoader(MainData,batch_size=constants.BACH_SIZE,shuffle=True)

def findLR(network):
  lr = constants.START_LR

  criterion = nn.CrossEntropyLoss()
  optimizer = optim.SGD(network.parameters(), lr=lr)
  lr_finder = LRFinder(network, optimizer, criterion)
  lr_finder.range_test(Myloader, end_lr=constants.END_LR, num_iter=100)

  lr_finder.plot()
  lr_finder.reset()

findLR(model)
```

Fonte: Produção da própria autora.

Quadro A5 - Arquivo myDataset.py

```
import pandas as pd
import torch
from torch.utils.data import Dataset
import torchvision.transforms.functional as TF
from PIL import Image

class MyDataset(Dataset):
  def __init__(self,pathname, classfile):
    df = pd.read_csv(classfile)
    self.pathname = pathname
    self.files = df["IMAGE\LABEL"]
    self.labels = torch.Tensor(df["zeis"]).type(torch.LongTensor)

  def __len__(self):
    return self.labels.size(0)

  def __getitem__(self, item):
    image = Image.open(self.pathname+"/"+self.files[item])
    x = TF.to_tensor(image)
    return x, self.labels[item]
```

Fonte: Produção da própria autora.

Quadro A6 - Arquivo myNetwork.py

```
import torchvision.models as models
import torch.nn as nn

# Rede de origem do transfer learning
model = models.resnet50(pretrained=True)
# model = models.resnet101(pretrained=True)
# model = models.resnet152(pretrained=True)

# Congelar as camadas de extração de características
for param in model.parameters():
  param.require_grad = False

# Substituir as camadas totalmente conectadas
fc = nn.Sequential(
  nn.Linear(2048, 1000),
  nn.ReLU(),
  nn.Dropout(0.4),
  nn.Linear(1000, 500),
  nn.ReLU(),
  nn.Linear(500,2),
  nn.ReLU(),
)

model.classifier = fc
```

Fonte: Produção da própria autora.

Quadro A7 - Arquivo EarlyStop.py

```
class EarlyStopping():
  def __init__(self, patience=5, min_delta=0):
    self.patience = patience
    self.min_delta = min_delta
    self.counter = 0
    self.last_epoch = 0
    self.best_loss = None
    self.early_stop = False

  def __call__(self, val_loss):
    if self.best_loss == None:
      self.best_loss = val_loss
    elif self.best_loss - val_loss > self.min_delta:
      self.counter = 0
      self.best_loss = val_loss
    elif self.best_loss - val_loss < self.min_delta:
      self.counter += 1

      print(f"INFO: Early stopping counter {self.counter} of {self.patience}")

      if self.counter >= self.patience:
        print('INFO: Early stopping')
        self.early_stop = True
```

Fonte: Produção da própria autora.

Quadro A8 - Arquivo constanst.py

```
VALIDATION_CLASS_FILE="artifacts/classes_validation.csv"
TEST_CLASS_FILE="artifacts/classes_test.csv"
TRAIN_CLASS_FILE="artifacts/classes_train.csv"

VALIDATION_DATASET="images/dataset"
TEST_DATASET="images/dataset"
TRAIN_DATASET="images/dataset"

MODEL_FILE="artifacts/model.pth"

# Hyperparams
BACH_SIZE=32
EPOCHS=50
LR=5.34E-03
```

Fonte: Produção da própria autora.