Projeto de Graduação

DEE DEPARTAMENTO DE ENGENHARIA ELÉTRICA

9 de Agosto de 2021

Segmentação Semântica de Áreas de Desmatamento na Amazônia a partir de Imagens Orbitais

Thiago Matheus Bruno da Silva

PUC RIO

Projeto de Graduação

DEE DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Segmentação Semântica de Áreas de Desmatamento na Amazônia a partir de Imagens Orbitais

Aluno: Thiago Matheus Bruno da Silva

Orientador: Raul Queiroz Feitosa

Corientador: Mabel Ximena Ortega Adarme

Trabalho apresentado com requisito parcial à conclusão do curso de Engenharia Elétrica na Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

Agradecimentos

Gostaria de agradecer aos meus pais e especialmente à minha mãe que me proporcionou estudar na PUC-Rio e sempre esteve ao meu lado nos momentos mais difíceis ao longo de toda minha trajetória nesta universidade se sacrificando de todos os jeitos para me dar uma boa educação. Devo muita gratidão aos meu orientador Raul Feitosa e coorientadora Mabel Ortega que me aceitarem de bom grado e confiaram em minha capacidade mesmo não me conhecendo tão bem para realizar esse projeto de grande relevância para à comunidade científica e ambiental brasileira. Também sou grato ao LVC (Laboratório de Visão Computacional) por ter me proporcionado infra-estrutura para realizar o projeto me cedendo computadores de alta performance para rodar os algoritmos desses trabalho.

Segmentação Semântica em áreas de Desmatamento

Resumo

Desmatamento é sem dúvida um enorme problema que afeta diretamente alguns desastres climáticos como a redução de biodiversidade, mudanças climáticas, entre outros. Portanto, é de tremenda importância detectar desmatamento recente. Motivado por este problema, este trabalho propõe um novo método para detecção automática de desmatamento baseado em segmentação semântica, utilizando a rede multitarefa ResUnet-a com uma nova tarefa semi-supervisionada baseado no algoritmo de *Change Vector Analysis* (CVA). O objetivo deste trabalho é estudar a contribuição do CVA no nosso problema de detecção de mudanças e comparar sua relevância com as outras tarefas. ALém disso, queremos observar as diferenças escolhas de threshold para o CVA usadas no treinamentoe seu impacto na tarefa principal de segmentação semântica. O método foi avaliado na Amazônia Legal, no Brasil. Nos nossos experimentos foram usados duas imagens do satélite Landsat 8, adquiridas em 2018 e 2019 as quais foram concatenadas na entrada da rede neural.

Palavras-chave: Aprendizado profundo, redes neurais, segmentação semântica, Aprendizado multitarefa, Desmatamento, Amazônia

Semantic Segmentation in Deforestation areas

Abstract

Deforestation is of no doubt an hugely important problem, which affects directly some destructive phenomena such as biodiversity reduction, climate change among other destructive phenomena. Therefore, is of tremendously importance to detect early deforestation. Motivated by this problem, this work proposes an new method for automatic deforestation detection, based on semantic segmentation, using ResUnet-a multitasking with a new Semi-Supervised Change vector analysis (CVA) task. The objective of this work is to study the contribution of CVA to our change detection problem and compare its relevance with the other tasks. Besides, we want observe the different choices of threshold used on the whole training and its impact on final main segmentation task. The method was evaluated in a region of the Brazilian Legal Amazon. In our experiments were used two images of Landsat 8 acquired in 2018 and 2019 which were concatenated on the input.

Keywords: Deep learning, Neural networks, Semantic Segmentation, Multitasking learning, Deforastation, Amazon

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

a Contextualização: O problema do desmatamento

A amazônia ou floresta amazônica é a maior floresta do mundo e possui cerca de 6 milhões km^2 [2] sendo 60 % pertencente ao Brasil. Ela é o lar de cerca 3 milhões de espécies, sendo 2,5 milhões (um terço de todas as árvores tropicais no planeta) de árvores aproximadamente [3]. Estas espécies estão sendo extintas muito rápidas com o passar dos anos, chegando a velocidade de mil vezes mais rápido que o processo natural [3]. Ainda, elas são de extrema importância para regular o ecossistema global e, também, descobrir novos remédios [4], [5].

Esta redução de biodiversidade é normalmente causada por grilagem de terras, mineração, expansão de agricultura industrial, exploração de madeira, pecuária e muitos outros. Logo, é claro a necessidade em controlar e monitorar a floresta.

Devido a este grande problema o Instituto Nacional de Pesquisas Espaciais (INPE) desenvolveu um projeto de monitoramento para a Amazônia Legal no Brasil: Projeto de Monitoramento do Desmatamento na Amazônia Brasileira (PRODES). Este projeto é responsável pelo monitoramento desta parte da Amazônia desde 1988 [Figuras 2 e 1]. Contudo ainda hoje em dia, a maioria dos métodos são manuais e requerem muito trabalho e tempo para detectar o desmatamento, tornando praticamente impossível detectar o desmatamento recente com rapidez. Neste trabalho, temos como objetivo propor um novo método automático para a detecção de desmatamento recente.

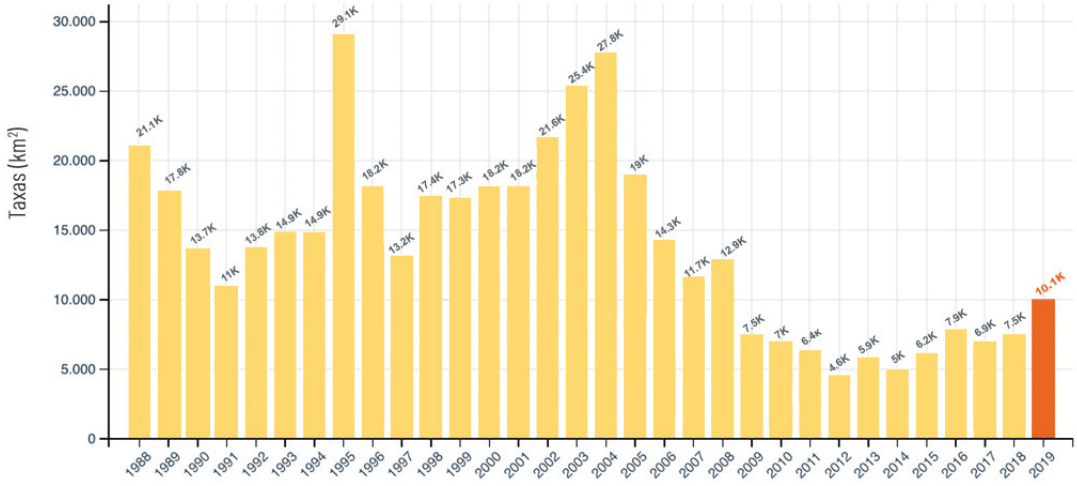


Figura 1: Histórico de desmatamento na Amazônia, mapeado pelo projeto PRODES, do INPE [1]

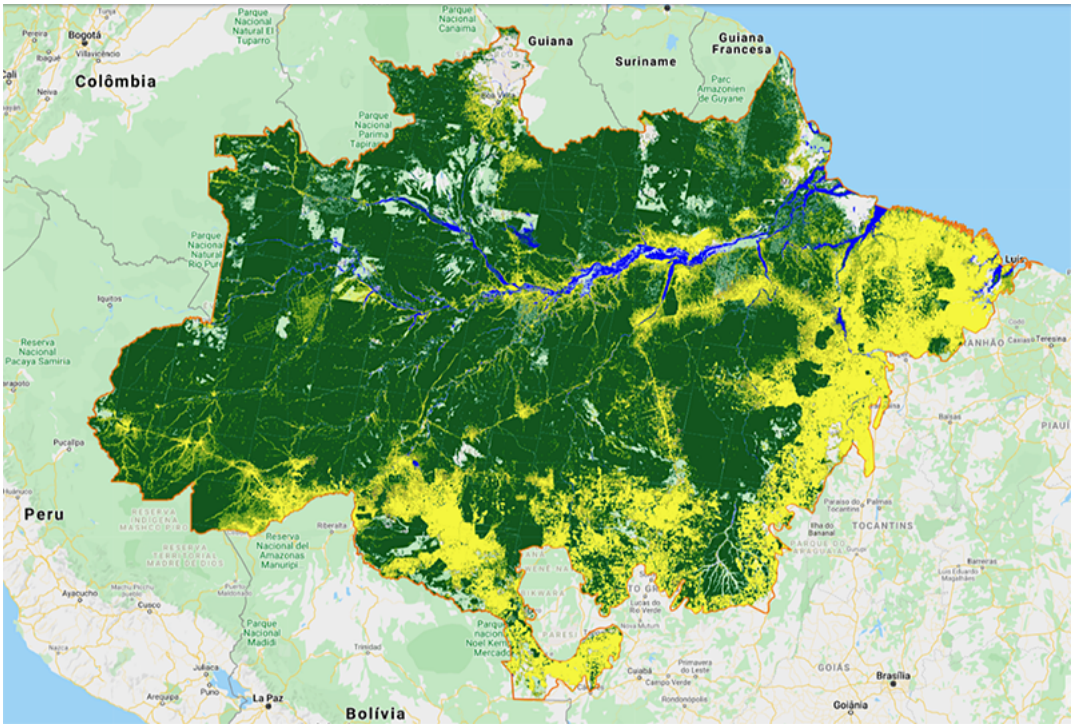


Figura 2: Área desmatada desde 1988 até 2019. Em verde área de floresta e amarelo desmatada [1]

b Trabalhos Relacionados

A literatura convencional das técnicas de detecção de mudanças (*change detection*), tais como *principal component analysis* (PCA), diferenças de imagem (*image differencing*), *change vector analysis* (CVA), foram usadas para detectar em imagens multi-temporais [6], [7]. Entretanto, usadas sozinhas, foi provado não serem tão efetivas nas detecções das mudanças e muito dependente da escolha de limiares (*thresholds*) para determinar quando ocorre ou não uma mudança [7]. Em particular, verificou-se que o CVA é uma boa escolha para nosso problerma de detecção de mudanças em zonas de vegetação [7], [6], devido às específicas bandas selecionadas para o algoritmo. *Change Vector Analysis* é um algoritmo de detecção de mudanças baseado majoritariamente em diferença de imagens. O pixel mudado é representado pela magnitude, a qual providencia informação sobre a intensidade da mudança, e uma fase, a qual fornece informação sobre o tipo de mudança. Neste trabalho iremos utilizar apenas a magnitude.

Com o avanço do aprendizado profundo, novas técnicas foram propostas para resolver o problema de detecção de mudanças. Alguns trabalhos mostraram grande potencial no uso de com redes neurais convolucionais para áreas urbanas e cobertas com terra: [8] propôs duas abordagens, uma utilizando o método classificação do pixel central e *early fusion* (concatenando duas imagens em anos diferentes) e outra utilizando redes siamesas; [9] utilizou redes siamesas com a *weighted contrastive loss*; [10] também utilizou redes siamesas e pseudo-siamesas com ramos para os pixels centrais e rodeados à esses pixels. Outros trabalhos também utilizaram redes convolucionais focados com classificação de áreas de vegetação como [11] usou redes siamesas para classificar o pixel central, [12] usou VGG16 para classificar retalhos de tamanho 128x128, [13] usou redes siamesas com a *contrastive loss*. Inspirado por estes trabalhos, decidimos seguir com uma abordagem de aprendizado profundo usando segmentação semântica, porém com uma rede estado-da-arte: ResUnet-a [14]. Esta rede segue a estrutura da ResUnet [15], uma arquitetura de codificação e decodificação (*encoder-decoder*) com conexões entre as estruturas de encoder e decoder, chamadas de *skip connections* [16], e também blocos convolucionais com conexões residuais herdadas da rede ResNet [17]. Juntamente aos blocos residuais, sua maior contribuição foi a adição de convoluções dilatadas a estes blocos e duas camadas de pooling em várias escalas denominadas de *pyramid scene sparse pooling (PSP pooling)*, localizados no meio e final da rede neural. Além dessa etapa

de codificação e decodificação, a rede utiliza também um aprendizado multitarefa com sub tarefas, calculadas com algoritmos tradicionais de visão computacional, para auxiliar a tarefa principal de segmentação semântica.

Inspirado por essa abordagem multitarefa e os algoritmos tradicionais de *change detection*, decidimos adicionar o algoritmo de *Change Vector Analysis* como mais uma tarefa do aprendizado da rede neural para colaborar com a tarefa principal de segmentação de áreas desmatadas. Nós escolhemos o CVA dentre os outros algoritmos, pois ele é uma ótima opção para o uso em áreas com vegetação [18], [7].

2 Revisão de conceitos

a Segmentação Semântica

Dentro de visão computacional existem três métodos básicos para analisar uma imagem. São eles:

(a) Classificação: Classifica a imagem toda em uma classe, como por exemplo: "Pessoa", "Gato", "Cachorro"

(b) Detecção de objetos: Localiza e classifica as classes em uma imagem desenhando um retângulo envolta do objeto desejado como: "Pessoa", "Animais", "Semáforos"

(c) Segmentação: Parecido com a ideia de classificar uma imagem inteira, desta vez o método associa o pixel da imagem à uma determinada classe. Entende exatamente, qual parte da imagem corresponde à uma determinada classe.

Na figura 3 podemos visualizar a diferença entre os métodos.

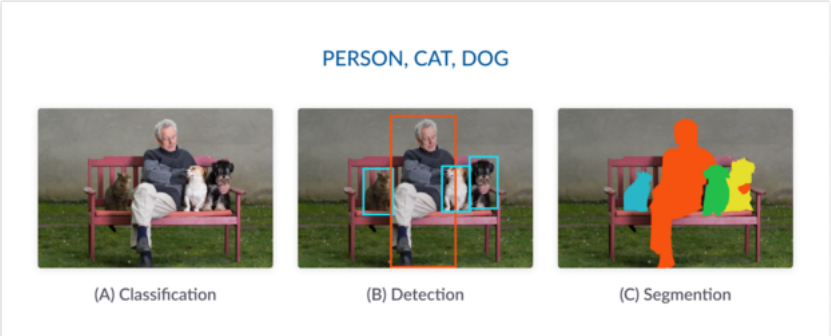


Figura 3: Métodos clássicos de visão computacional.

Neste trabalho será utilizado o método de segmentação semântica, o qual é um método particular de segmentação que tem como objetivo assimilar o pixel a uma determinada classe sem diferenciar as entidades dentro de uma mesma classe, como pode ser visto na figura 4

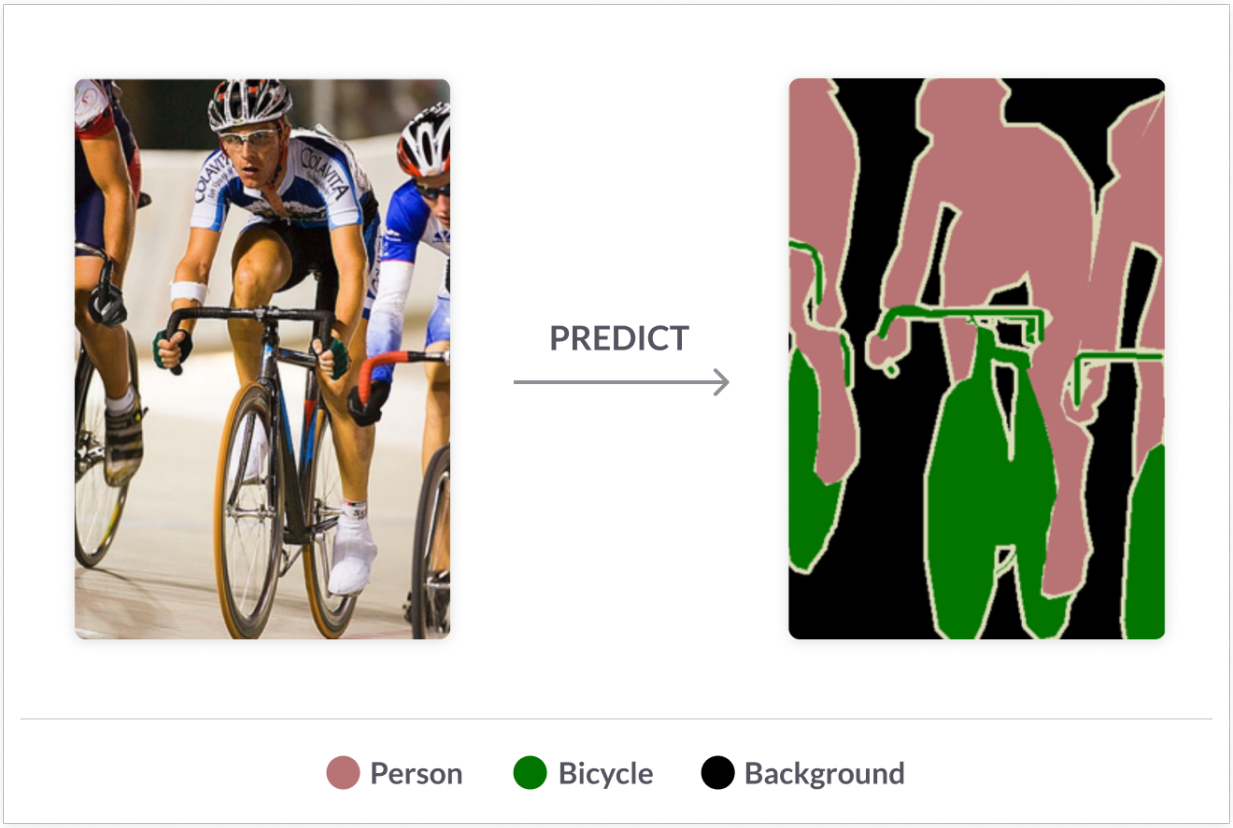


Figura 4: Exemplo de Segmentação Semântica

b Aprendizado Profundo

Aprendizado profundo, também conhecido como *deep learning*, é um método de aprendizado de máquina, também conhecido como *machine learning*, baseado em redes neurais capaz de conseguir modelar problemas complexos, lineares e não lineares, utilizando um grande volume de dados. Redes neurais é um algoritmo de aprendizado de máquina inventado na década de 50 que tinha como objetivo simular a plasticidade neural do cérebro, capacidade do cérebro em desenvolver novas conexões sinápticas entre os neurônios a partir da experiência e do comportamento do indivíduo. Para isso o algoritmo se baseia na estrutura dos neurônios e na comunicação sináptica entre eles.

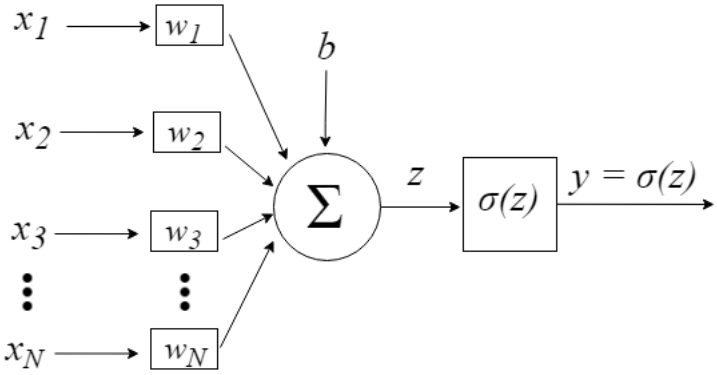


Figura 5: Perceptron. $x_i \in x_1, ..., x_N$ é o sinal de entrada, b é o bias e σ é a função de ativação.

$$z = \sum_{i}^{n} (w_i x_i + b) \tag{1}$$

Na figura 5 podemos ver um exemplo do modelo matemático de um neurônio chamado de *perceptron*, criado em 1958 por Frank Rosenblat. A ideia do neurônio artificial é receber um sinal numérico de entrada assim como um sinal elétrico chega pelos dendritos de um neurônio biológico. Após essa etapa, o neurônio biológico processa esses sinais elétricos e decide se o sinal deve continuar sendo propagado ou não para o próximo neurônio. Analogamente ao modelo biológico, o neurônio artificial também processa os dados de entrada multiplicando cada uma das entradas por um peso w_i [equação 1] e somando com um bias. Em seguida, esse somatório ponderado entra numa função de ativação σ, como por exemplo o degrau unitário [Figura 6] que ativa a entrada caso ela seja maior que zero, ou anula ela caso contrário.

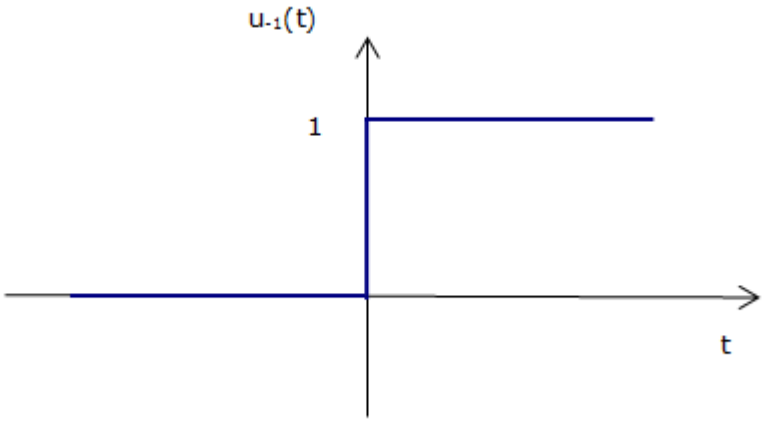


Figura 6: Função degrau unitário

A ideia do perceptron é aproximar uma certa função f^*, como por exemplo um certo classificador $y = f^*(x)$ o qual mapeia uma entrada x a uma categoria y. O perceptron define um mapeamento $y = f(x; \theta)$, o qual aprende os valores dos parâmetros θ os quais resultam numa melhor aproximação da função f^*. Olhando para apenas um único perceptron, em um problema de classificação com duas entradas apenas (coordenadas x e y de um ponto no plano cartesiano R^2) por exemplo, podemos ver que o que o algoritmo faz é procurar uma reta que separe corretamente um determinado conjunto de dados. Entretanto, isso não é o suficiente para resolver problemas mais complexas, visto que nem todos os prolemas de classificação podem ser resolvidos por uma única reta.

Para resolver problemas mais complexos, adicionou-se várias camadas intermediárias, entre as entradas e saídas, obtendo o que nós conhecemos perceptron de multi camadas, também conhecido como *MLP: multi layer perceptron* [Figura 7]. Assim, aumentou-se os número de parâmetros da rede sendo possível aproximar funções mais complexas. Olhando de novo para o exemplo de pontos no plano cartesiano R^2, podemos entender esse aumento da complexidade como tentar classificar vários grupos de pontos. No caso de três grupos, já não seria possível separar os pontos com uma reta somente. Ao se inserir mais perceptrons na primeira camada o que ocorre é a adição de retas à separação dos pontos. Além da adição de mais neurônios, também é utilizado funções de ativação não-lineares em cada um deles, como a ReLU, sigmoid, softmax, e tangente hiperbólicas. O que ocorre é a transformação dessas retas em curvas mais complexas. Sendo assim, ao adicionarmos infinitos perceptrons com funções de ativações não lineares é possível resolver problemas muito complexos, ou seja, aproximar funções não lineares.

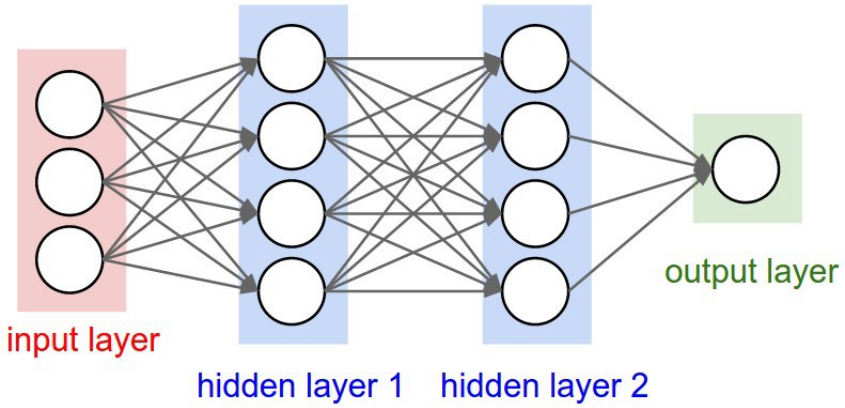


Figura 7: Rede neural perceptron multi camadas

Atualmente a função mais usada é a ReLU devido a sua performance e simplicidade, principalmente nos cálculos dos gradientes. Outra função de ativação muito usada é a *softmax* [Equação 2]. Esta função de ativação é utilizada normalmente na última camada para separar as probabilidades de saída, por exemplo, em um algoritmo de classificação. O motivo de ser muito usada como a função de ativação da última camada é devido a sua expressão matemática fornecer probabilidades normalizadas. Sendo assim, a soma de todas as probabilidades deve ser igual a 1, visto que sempre dividimos pela soma dos valores do vetor $z = W^T h + b$. Deste modo, sempre teremos um valor máximo, que induz o algoritmo a separar as classes.

$$p_i = softmax(\mathbf{z})_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{2}$$

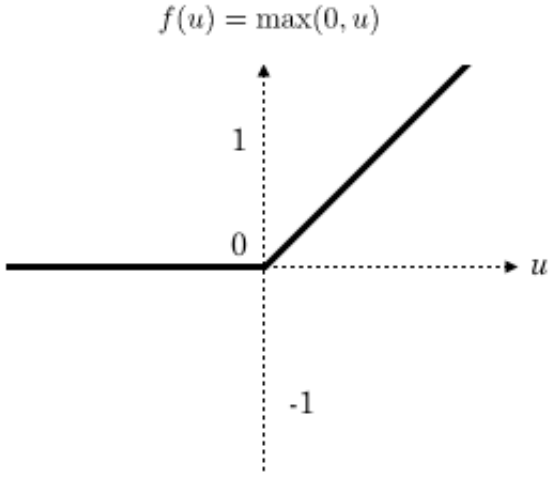


Figura 8: Gráfico e fórmula da função ReLU (Rectified Linear Unit)

Devido ao grande número de camadas, começa a se tornar difícil a convergência do algoritmo e novos métodos começam a ser necessários para otimizá-lo como o descida de gradiente ou *gradient descent*. Assim, foi dado início ao campo que conhecemos hoje como aprendizado profundo pela profundidade de camadas que esses algoritmos usam. Esse maior número de camadas resulta num maior número de parâmetros no algoritmo e portanto maior número de operações. Isso faz com que os resultados sejam muito mais precisos, porém é necessário uma grande quantidade de dados e poder computacional, que não era expressivo como nos dias de hoje e nem de fácil acesso para pessoas que não trabalhavam em um laboratório de pesquisa ou grande empresa.

Devido aos novos algoritmos de otimização como o gradiente descendente, o qual falaremos na seção 2.c, foi possível inserir novas camadas entre a entrada e saída, chamadas assim de camadas escondidas (*hidden layers*).

Apesar do poder computacional ter aumentado muito ao longo dos últimos anos o método perceptron multi camadas ainda demanda muito poder computacional e quando começamos a trabalhar com dados não estruturados como imagens e áudio começa a se tornar inviável utilizá-lo. Visto isso, criou-se um novo método de aprendizado profundo, as chamadas redes convolucionais.

c Descida de gradiente

Para guiarmos a convergência do nosso processo de aprendizado de uma rede neural utilizamos o método descida de gradiente ou *gradient descent*. Ele é um método de otimização, que tem como princípio maximizar ou minimizar uma função $f(x)$, alterando o valor de de x a cada iteração. No âmbito de redes neurais, queremos normalmente minimizar o valor da nossa função de custo ou função de erro, já que esperamos uma melhor performance do algoritmo quando o erro tiver o valor 0.

Para achar o mínimo da função, o método consiste em achar a direção a qual a função decresce mais rapidamente, em outras palavras, a direção onde a derivada é mais negativa. Para compreender melhor, podemos olhar para uma direção qualquer de uma função de várias variáveis $f : \mathbb{R}^n \to \mathbb{R}$ e investigar sua derivada direcional em uma direção $u \mid ||u||_2 = 1$.

$$\begin{aligned} min_u &= u^\top \nabla_x f(x) \\ min_u &= ||u||_2 ||\nabla_x f(x)||_2 cos\theta \end{aligned} \tag{3}$$

onde θ é o ângulo entre o vetor unitário u e o vetor gradiente. Sabendo que $||u||$ possui a norma igual a 1 e que o vetor gradiente não depende do vetor direção u, podemos restringir a $min_u = cos\theta$. Portanto, a derivada direcional possui valor mínimo, quando o vetor u está na direção oposta do gradiente, ou seja, $\theta = 180^o \to cos(180^o) = -1$. Isso quer dizer que essa é a direção onde a função f é mais decrescente, ou seja, a derivada é mais negativa. Assim, sabendo o caminho mais decrescente podemos atualizar o ponto em questão $x^{'}$ [19] [Equação 7] .

$$x' = x - \epsilon \nabla_x f(x) \tag{4}$$

Sendo ϵ a taxa de aprendizagem do algoritmo. A taxa de aprendizagem é um escalar positivo que serve para ajustar os passos que o algoritmo dá em direção ao mínimo global. Uma vez que ele o algoritmo pode acabar convergindo para mínimos locais e ficar estagnado nessas posições [Figura 9].

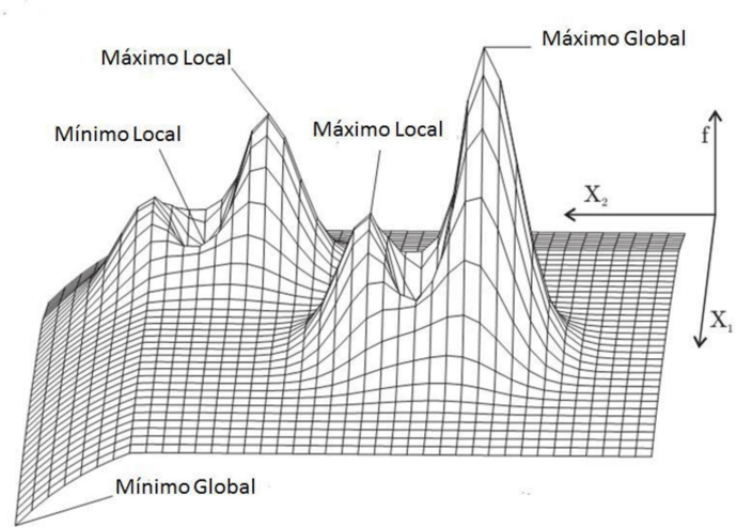


Figura 9: Máximos e mínimos de uma função

A ideia do algoritmo pode ser melhor compreendida visualizando o problema com uma função de uma variável apenas [Figura 10].

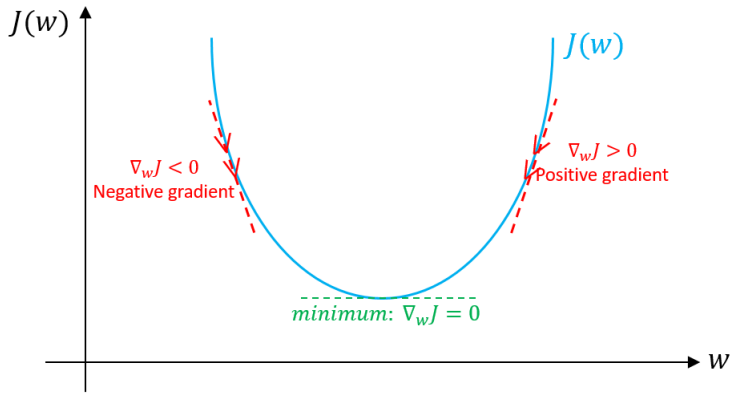


Figura 10: Exemplo do método descida de gradiente aplicado à uma função de uma variável

No caso de aprendizado de máquina, temos como objetivo minimizar a nossa função de custo ou função de erro, a qual irá guiar o nosso aprendizado. No geral, algoritmos de machine learning necessitam de um grande conjunto de dados $\mathbb{D} = \{x^{(1)}, ..., x^{(m)}\}$ para treinar o algoritmo, o qual é computacionalmente caro. Desta forma, podemos calcular o gradiente de cada iteração (época) do nosso treinamento como a média do gradiente da função de custo para cada amostra do nosso junto de dados [19] [Equação 6].

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L_\theta(x^{(i)}, y^{(i)}, \theta) \tag{5}$$

Onde J é a função de custo total e L é a função de custo por amostra, $x^{(i)}$ é um dado da amostra $\mathbb{D}$ e $y^{(i)}$ sua referência.

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L_\theta(x^{(i)}, y^{(i)}, \theta) \tag{6}$$

Assim, podemos ver que a atualização dos parâmetros θ é dada por:

$$\theta' = \theta - \epsilon \nabla_\theta J(\theta) \tag{7}$$

Como sabemos que a função J calcula o erro na saída da nossa rede com base na referência associada a saída do modelo $g(x;\theta)$. Entretanto, ao meio da arquitetura de uma rede neural não possuímos uma referência explicita para cada saída e cada camada da rede e, portanto, não temos como calcular o erro de cada saída explicitamente. Para solucionar o problema, calculamos a derivada com respeito ao parâmetro

Podemos ver na equação 6 que o gradiente só é calculado após percorrer todo o conjunto de dados $\mathbb{D}$. Isso torna todo o processo muito custoso computacionalmente, o qual possui complexidade de $\mathcal{O}(\frac{m}{d} log(1/\epsilon)$ [20], sendo d o número de parâmetros, m o tamanho do conjunto de dados e ϵ é o erro mínimo aceitado pelo algoritmo até a convergência **excess error** $J(\theta) - \mathbf{min}_\theta J(\theta) \leq \epsilon$. Isso se torna inviável quando a quantidade de dados é muito extensa, pois temos que esperar percorrer tudo para atualizar os pesos. Sendo assim, o que se fez na prática é implementar uma variação do método chamado de *Stochastic Gradient Descent* (SGD) o qual atualiza os pesos após **cada amostra** no conjunto de dados. O SGD possui complexidade $\mathcal{O}(\frac{d}{\epsilon})$ o que mostra mesmo a depêndencia de ϵ

sendo pior, já que é um número muito pequeno, notamos que para valores de m muito grandes, o SGD sai vitorioso nessa disputa computacional. A ideia é que como atualizamos sempre os pesos, o algoritmo irá convergir antes de passar pelo dataset inteiro, fazendo menos iterações. Entretanto, como estamos atualizando sempre o modelo para toda amostra, temos um maior ruído acrescentado o que pode levar a problemas de convergência. Sendo assim, o mais utilizado na prático é um método intermediário chamado de *Mini-Batch Gradient Descent*, o qual atualiza os parâmetros do modelo após um determinado conjuntos de samples chamados de **minibatch** $\mathbb{B} = \{x^{(1)}, ..., x^{(m')}\} \mid m' < m$. Assim, atualizando o modelo em minibatches temos uma maior aproximação para o conjunto de dados como um todo inserindo menos ruído, além de ter um custo computacional menor do que o tradicional GD. Na figura 11 podemos ver a diferença de convergência de cada uma dos métodos de descida de gradiente.

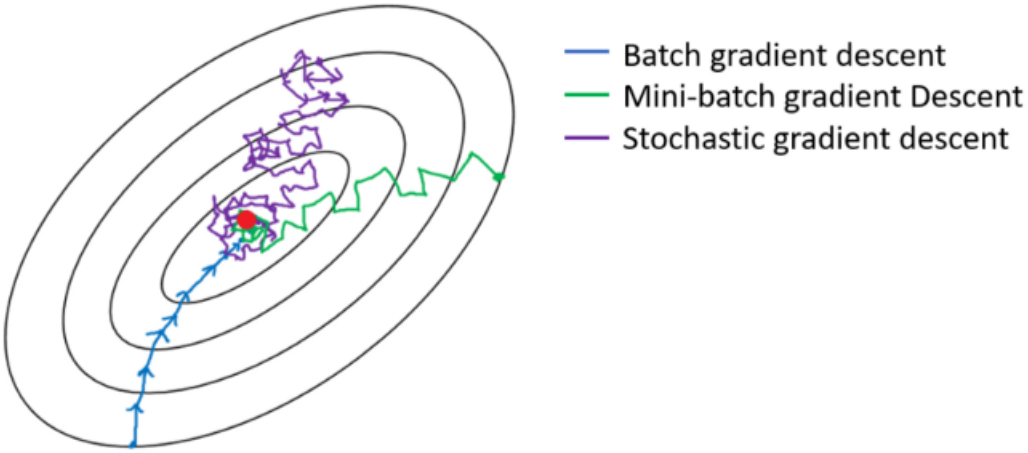


Figura 11: Tipos de descida de gradiente e suas convergências durante o treinamento

d Redes Neurais Convolucionais

As redes convolucionais, também conhecidas como *CNN's*, ganharam destaque em aplicações utilizando principalmente imagens. As operações de convolução substituem a multiplicação de matrizes usual de uma rede neural tradicional.

A convolução é uma operação realizada com duas funções reais. No nosso âmbito do processamento de um computaodr, essas funções viram discretas uma vez que nossos dados já estão amostrados e quantizados. Sendo assim, vamos podemos definir a operação de convolução para funções discretas como:

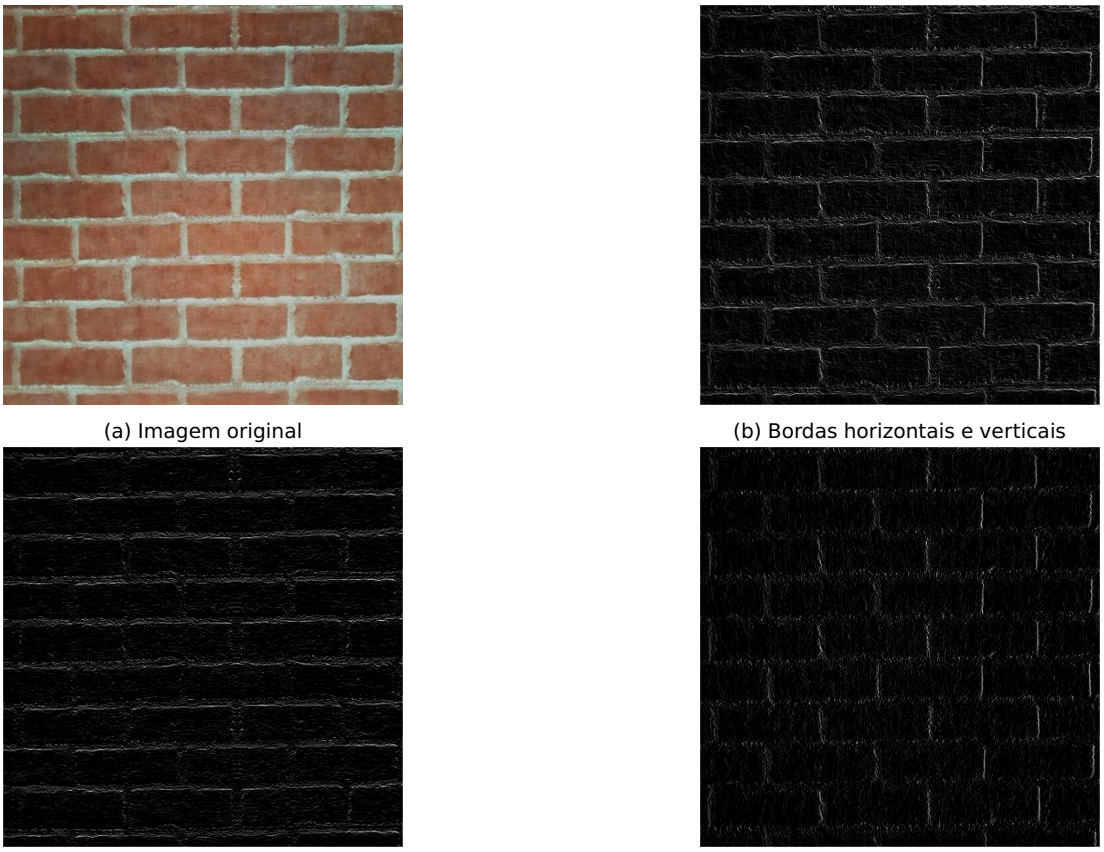
$$s(t) = (x * k)(t) = \sum_{a=-\infty}^{\infty} x(a)k(t-a) \tag{8}$$

Normalmente na literatura, chamamos $x(t)$ de sinal de entrada e $k(t)$ de kernel. Entretanto no nosso caso específico de visão computacional o nosso sinal de entrada, por consequência também o kernel, é uma Imagem vetor multidimensional, ou seja, um tensor.

No caso de uma imagem bidimensional, uma foto em tons de cinza, podemos definir a convolução como:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{9}$$

Essa operação serve para extrair informações do sinal de entrada. No caso de imagens, aplicando diferentes tipos de kernels obtemos diferentes tipos de informação extraídas da imagem. Um possível exemplo pode ser visto utilizando um filtro de detecção de bordas como o Sobel. Para extrair as bordas da imagem inteira utiliza-se dois filtros. Um para extrair as bordas na horizontal e o outro na vertical. Podemos ver o resultado das bordas na figura 12.



(a) Imagem original

(b) Bordas horizontais e verticais

(c) Bordas horizontais

(d) Bordas verticais

Figura 12: Filtro de Sobel aplicado à uma imagem de tijolos

Tendo em vista a operação de convolução, as redes convolucionais utilizam essa operação ao invés da multiplicação de matrizes utilizado nas redes neurais *feed foraward* tradicionais. Essa mudança ocorreu principalmente ao fato de ser muito custoso lidar com imagens utilizando multiplicação de matrizes, uma vez que para processar uma imagem com uma rede neural se deve "esticar"a imagem em um vetor coluna 1D e em seguida multiplicar todos os pixels por todos os nós da próxima camada. Já com a convolução conseguimos extrair a mesma informação porém com um custo computacional muito menor. Isso deve principalmente ao fato das convoluções possuirem uma conectividade esparsada [Figura 13] (*sparse connecvity*) e compartilhamentos de parâmetros [Figura 14] (*parameter sharing*). O primeiro, pode ser entendido analisando o campo receptivo do kernel. Em uma convolução, um pixel em uma camada de saída é somente afetado por um grupo dos pixels na entrada e não por todos os neurônios da camada anterior. O mesmo pode ser visto para um pixel na entrada de uma camada, o qual afeta apenas um conjunto de pixels da camada de saída, enquanto em uma rede totalmente conectada afetaria todos os neurônios. O compartilhamento de parâmetros por sua vez, se refere ao fato de durante a convolução você utilizar o mesmo filtro em toda imagem, durante o deslocamento espacial da operação de convolução, ao contrário de uma MLP a qual você possui um peso para cada pixel na entrada.

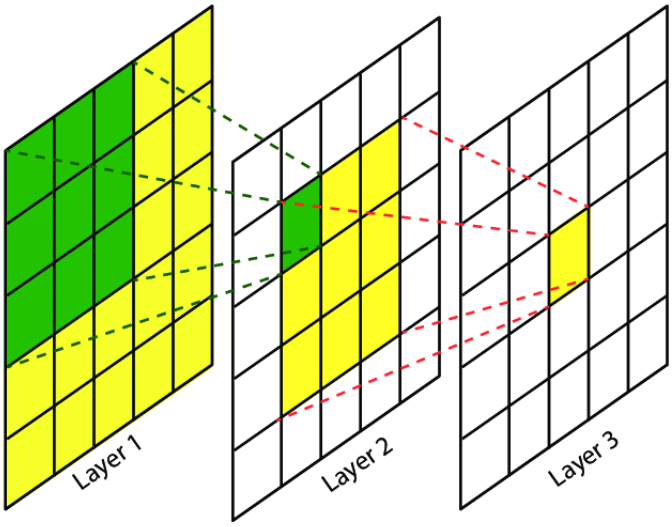


Figura 13: Campo receptivo de uma convolução

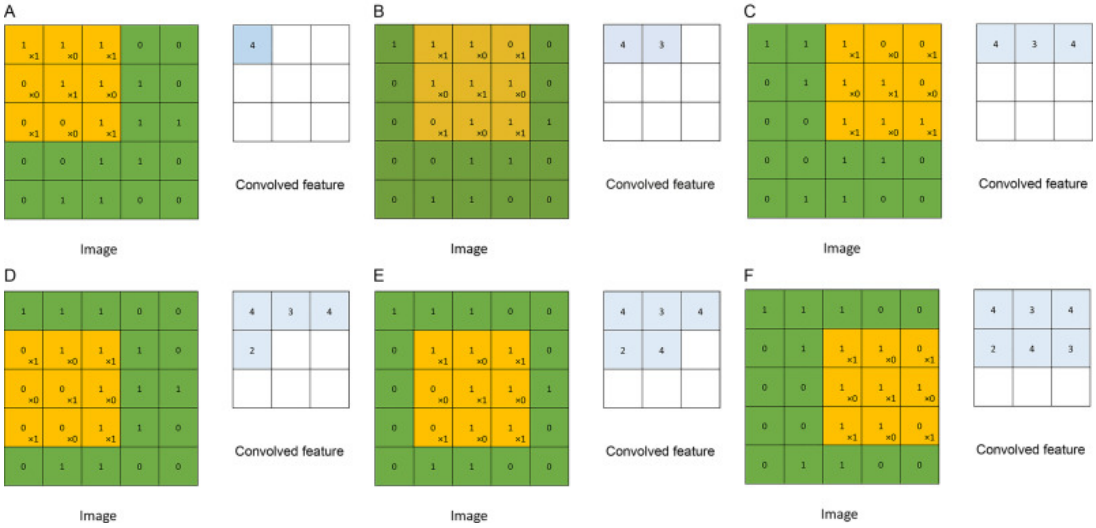


Figura 14: Compartilhamento de parâmetros

Durante o treinamento, a rede neural utiliza esses filtros convolucionais para extrair informações das imagens como foi mostrado na figura 12. Entretanto, diferentemente do filtro de Sobel que possui kernels pré-definidos, a rede neural aprende os parâmetros do kernel para aquele conjunto específico de dados. Sendo assim, a cada camada ela possui um determinado conjunto de filtros que são responsáveis por extrair várias características da imagem, por exemplo: filtros de bordas circulares, horizontais de determinadas cores. Após passar pela camada de convolução, o tensor de saída passa por uma função de ativação não linear assim como nas redes *feed forward*.

Outra caraterística fundamental das redes convolucionais é o uso de pooling. Visto que inferir sobre o tamanho original de uma imagem é muito custoso do ponto de vista computacional, a cada camada se reduz o tamanho da imagem e se acha os melhores filtros para aquele tensor já processado. O pooling é aplicado de maneira parecida com a convolução com um tamanho de kernel específico. O kernel vai deslizando sobre a imagem, proém ao invés de multiplicar o valor da matriz pelo kernel, pega o valor máximo daquela região, no caso do max pooling. Outras formas de pooling podem ser feitas, como a média dos valores (Average Pooling), porém o max pooling é o mais utilizado e menos custoso. Um exemplo de como é aplicado o max pooling pode ser visto na figura 15.

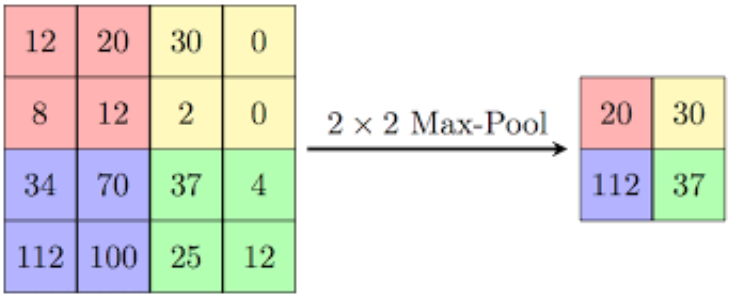


Figura 15: Max-pooling

Assim, consegue-se reduzir o custo e entender informações na imagem de diferentes escalas. Um exemplo de uma arquitetura básica de uma CNN pode ser visto na figura 16

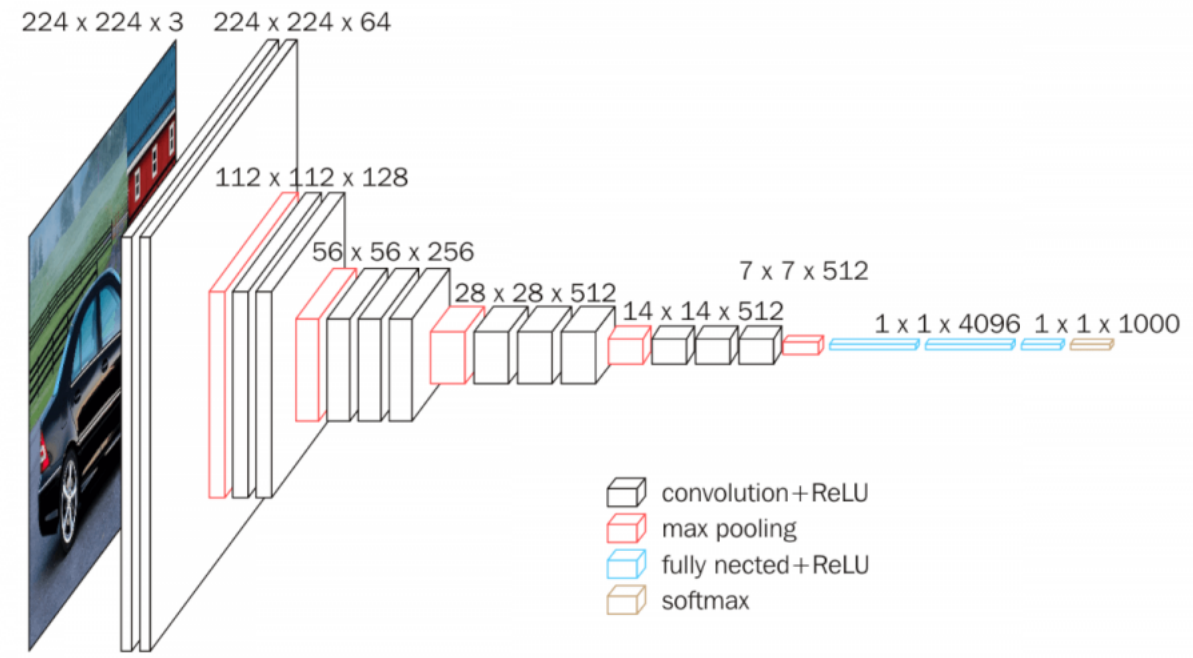


Figura 16: Arquitetura básica de uma CNN

1 ResUnet-a

Neste trabalho iremos utilizar uma rede convolucional específica de segmentação semnântica, a ResUnet-a. A ResUnet-a é principalmente baseada na U-Net [16], uma arquitetura clássica de segmentação semântica. Como foi explicado no começo deste capítulo, redes tradicionais convolucionais de classificação utilizam uma etapa de codificação, a qual é basicamente formada por camadas de convolução, pooling e funções de ativação não lineares. Sua segunda etapa é classificar a imagem de entrada com base do tensor codificado de dimensão reduzida. Como em segmentação semântica temos como objetivo classificar cada pixel da imagem de entrada, não podemos trabalhar usando apenas do novo tensor codificado. Para isso é realizado um sobre-amostragem (*Upsample*) visando recuperar o tamanho da imagem original. Algumas abordagens de sobre-amostragem podem ser usadas como convolução transposta, onde basicamente é uma camada convolucional que aprende a fazer uma sobre-amostragem mais adequada. No geral isso aumenta o número de parâmetros da rede tornando o treinamento mais caro computacionalmente. Por isso, normalmente usamos abordagens mais simples como a repetição de vizinhos próximos [Figura 17].

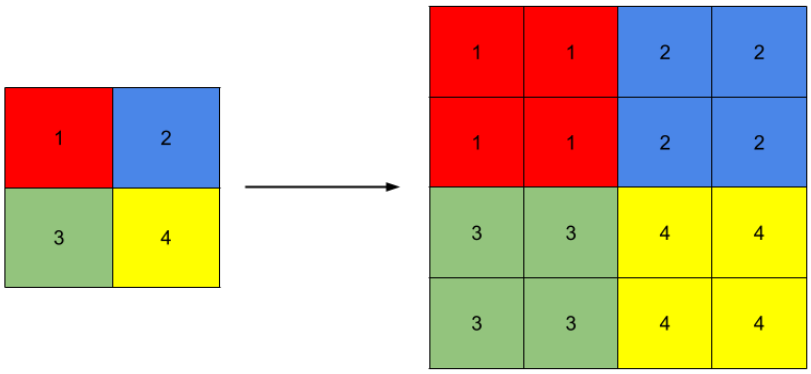


Figura 17: Exemplo de um método de sobre-amostragem utilizando vizinhos próximos

Sendo assim, ao invés de utilizar a convolução transposta, é muito comum utilizar uma camada de convolução, seguido de uma sobre-amostragem.

Além desta etapa no processo de decodificação, a rede U-Net propôs mais uma etapa para auxiliar o upsample, que seriam as chamadas *skip connections*. Essas conexões, são responsáveis por juntar *features* das camadas de codificação as camadas de decodifcação, concatenando os tensores das camadas de codificação juntamente com os da etapa de decodificação. Sendo assim, consegue-se recuperar o tamanho da imagem com maior precisão sem aumentar o número de parâmetros. A arquitetura da rede U-Net, com suas skip connections, pode ser vista na Figura 18.

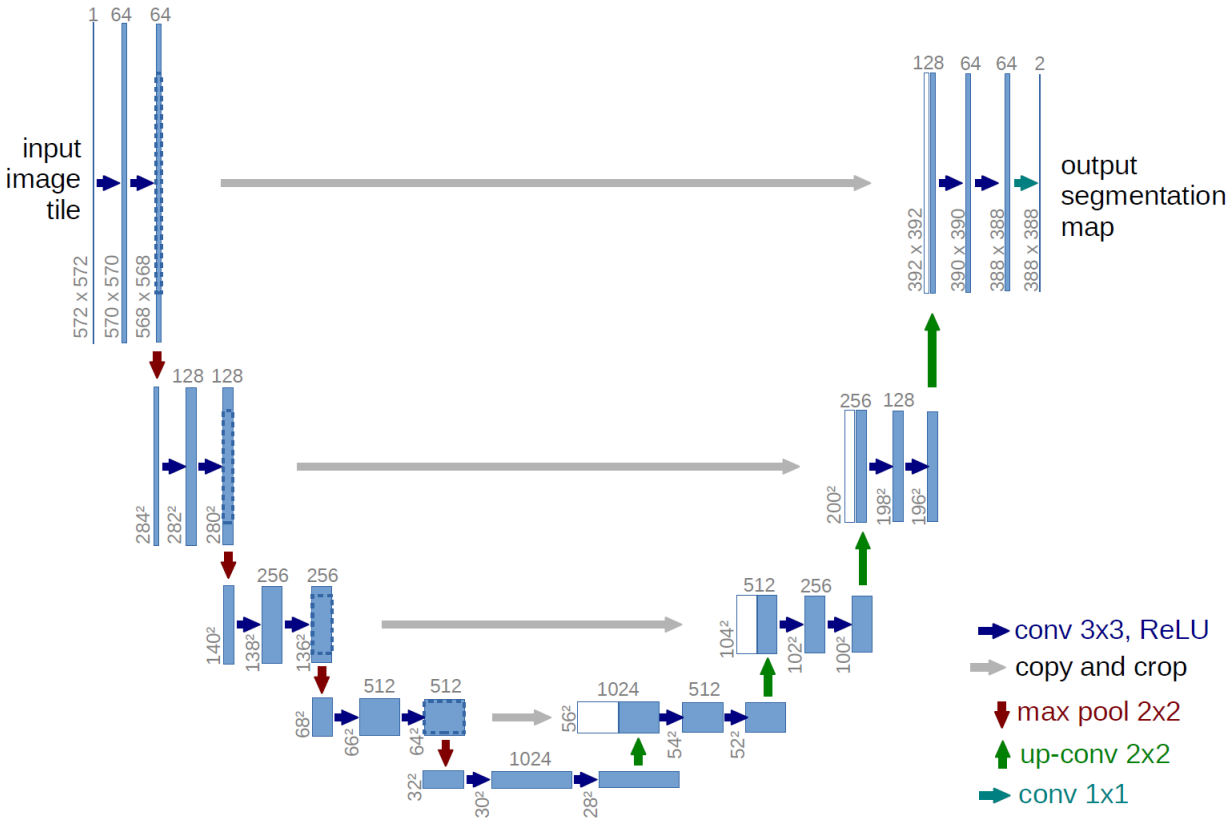


Figura 18: Arquitetura da rede U-Net

Outra rede que influenciou bastante a ResUnet-a, foi a chamada ResUnet [15]. Esta rede se utiliza da arquitetura da U-Net, entretanto suas camadas de convolução utilizam os blocos residuais propostos na ResNet [17] em 2015. Esses blocos são responsáveis por somar um termo de entrada da camada convolucional [Figura 19], pois faz com que a rede aprenda a transformação identidade, possibilitando uma rede mais profunda e com ganhos consideráveis comparados a redes com a mesma profundidade sem blocos residuais. A ideia por trás se baseia em aumentar a profundidade da rede sem alterar os cálculos e para isso precisamos da transformação identidade, a qual não é aprendida tão facilmente pelos nosso métodos de otimização com gradiente descendente e retro-propagação. Para termos de fato a transformação identidade precisamos que o resíduo aprendido seja zero ou muito próximo disso. Sendo assim, a rede irá aprender o necessário para predizer corretamente a saída e ao mesmo tempo zerar o resíduo $F(x)$ [19], uma vez que é uma solução ótima mais fácil que aprender direto a transformação identidade. Podemos observar a ideia da transformação identidade na figura 20.

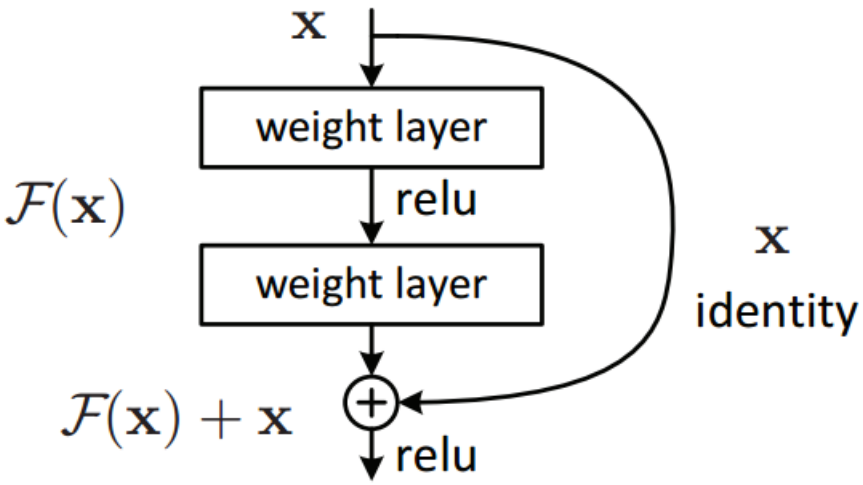


Figura 19: Blocos residuais da ResNet

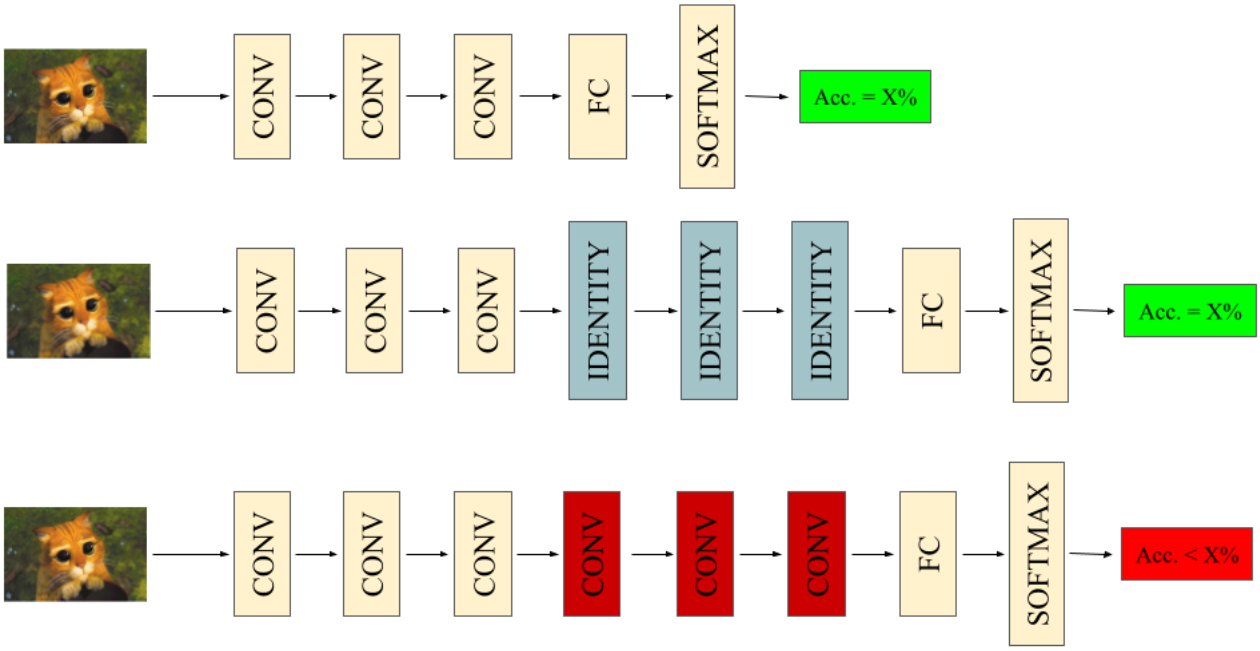


Figura 20: (Topo) Comparação dos resultados de uma arquitetura utilizando uma rede convolucional com outra com (Meio) a transformação identidade e com (fundo) a arquitetura básico com mais convoluções

Outra fato importante dos blocos residuais é que evitam que o gradiente seja zerado durante a retropropagação, fenômeno chamado de *vanishing gradient*. Isso pode ser provado observando a derivada de um bloco residual, pois agora somando o termo de entrada, teremos a constante 1 sempre se somando a derivada do resíduo, impossibilitando de zerar os gradientes. Podemos observar esse fato mais explicitamente olhando a fórmula da figura 22.

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)$$

(a) Formula de uma camada residual particular

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)$$

(b) Formula para várias camadas residuais

Figura 21: Formulas das camadas residuais em (a) uma camada e (b) várias camadas ao longo da rede neural a partir da l-ésima camada. $F(.)$ é a função que representa as camadas convolucionais

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right)$$

Figura 22: Gradiente de L camadas residuais a partir da l-ésima camada

Além dessas principais ideias herdadas de outras arquiteturas a ResUnet-a também adotou as chamadas convoluções dilatadas [Figura 23]. A ideia principal é adicionar a cada bloco convolucional, outras convoluções com taxas de convoluções diferentes extraindo características em diferentes tipos de escala. Sendo assim, em cada bloco convolucional, a rede iria ter uma camada de convolução sem dilatação e outras camadas para cada dilatação usada, todas em paralelo [Figura 30 (b)].

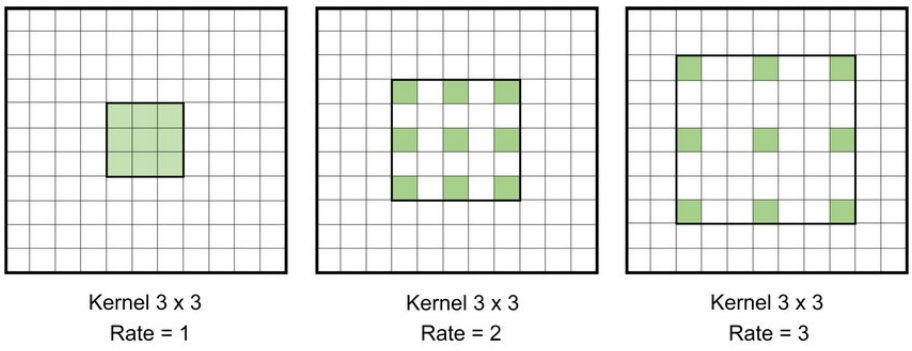


Figura 23: Exemplo de convoluções dilatadas

Outra grande diferença da ResUnet-a para as demais variações da U-Net, é a introdução de um *Pyramid Scene Parsing pooling* (PSP pooling) [Figura 30 (c)]. Essa camada foi introduzida no meio e final da arquitetura, objetivando extrair ainda mais características com poolings em diferente escalas seguidos de uma camada convolucional. A ideia dessa camadas é aumentar a extração de informação do contexto do fundo (*background*) da imagem de entrada. Após os múltiplos poolings, cada uma das saídas em diferentes escalas são concatenadas e passadas para a próxima camada da rede.

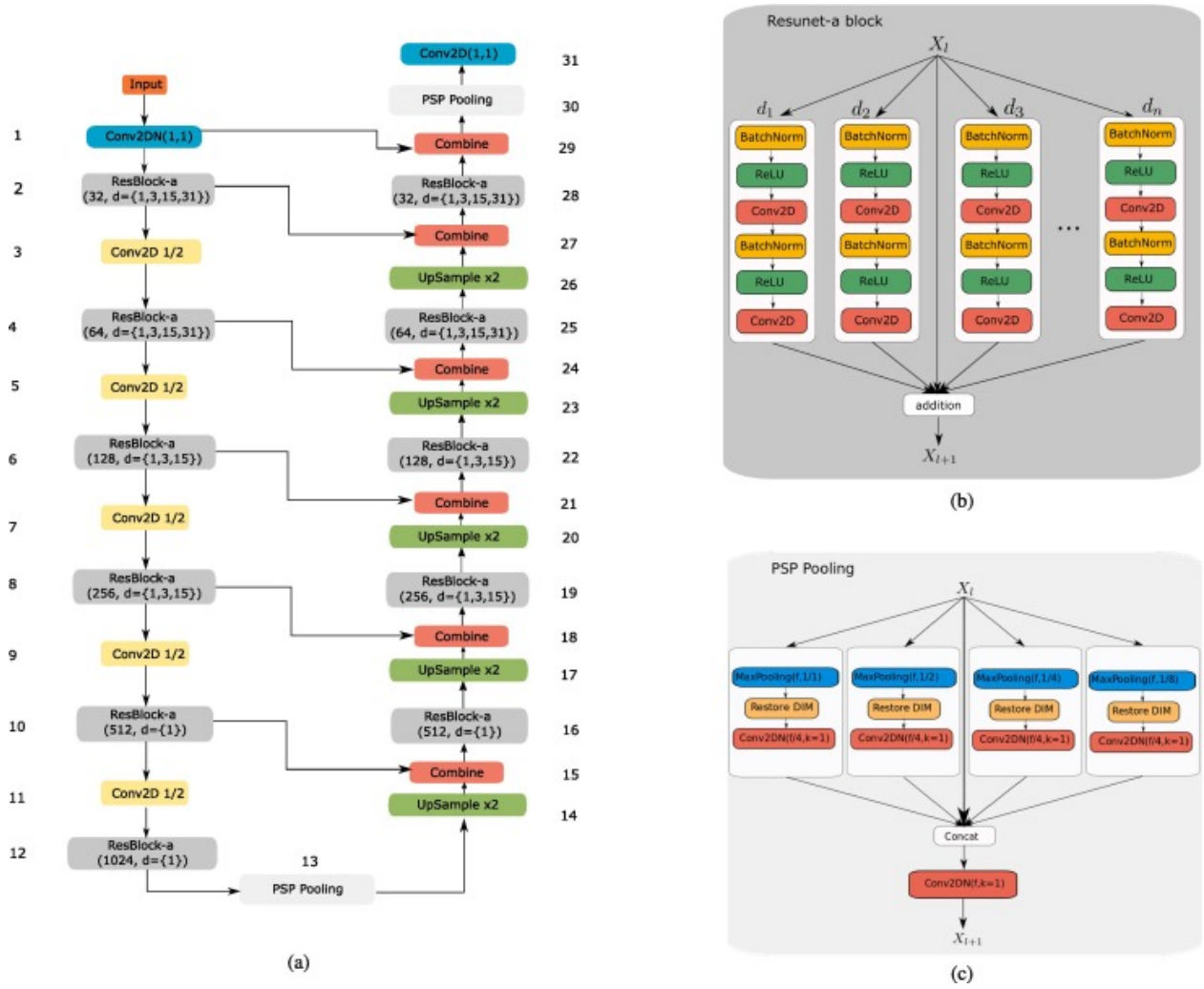


Figura 24: (a) Arquitetura da ResUnet-a. Na esquerda se encontra (descendente) a parte de codificação. Na direita (ascendente) a parte de decodificação. No meio e final podemos ver as camadas de PSP pooling. A última camada de classificação possui o número de canais equivalentes ao número de classes do dataset. (b) Blocos convolucionais da ResUnet-a. Todas as convoluções possuem os mesmos número de filtros. No caso $d_1, ..., d_n$ são as taxas de dilatação das convoluções. (c) Composição da camada de Pyramid scene parsing pooling. Na imagem, o pooling desenhado é específico para uma imagem de entrada de tamanho 256.

Outra contribuição, além da arquitetura principal foi o uso de um aprendizado multitarefa para auxiliar a tarefa principal de segmentação semântica. Além dela, outras três tarefas foram utilizadas: *Boundary Detection*, *Distance Transform* e *HSV color transform*. A ideia de usar essas outras tarefas complementares é cada uma delas oferecer um maior detalhamento das máscaras geradas da segmentação semântica. A transformação distância fornece uma conectividade topológica das máscaras de segmentação, tal como a extensão dos objetos na imagem. A detecção de bordas, por sua vez, ajuda num melhor entendimento da extensão das máscaras de segmentação semântica. A transformação do espaço de cor em HSV providencia informação adicional entre a variação de cor e a também a extensão do objeto. Isso ajuda as predições do modelo a manterem "vivas"as informações dos pequenos detalhes da imagem original até a última camada de segmentação semântica.

Todas essas tarefas são operações simples de visão computacional clássica que visam auxiliar o resultado da tarefa de classificação de pixel. Outro grande contribuição é que a geração de labels é feita a partir de rotinas tradicionais como, por exemplo utilizando a bibilioteca **OpenCV**, a partir das referências da segmentação semântica. Portanto, além de melhorar a performance da classificação do modelo a dificuldade para se obter as novas referências dos novos aprendizados é praticamente

nula. Um exemplo do predição de cada uma das tarefas pode ser vista na Figura 25

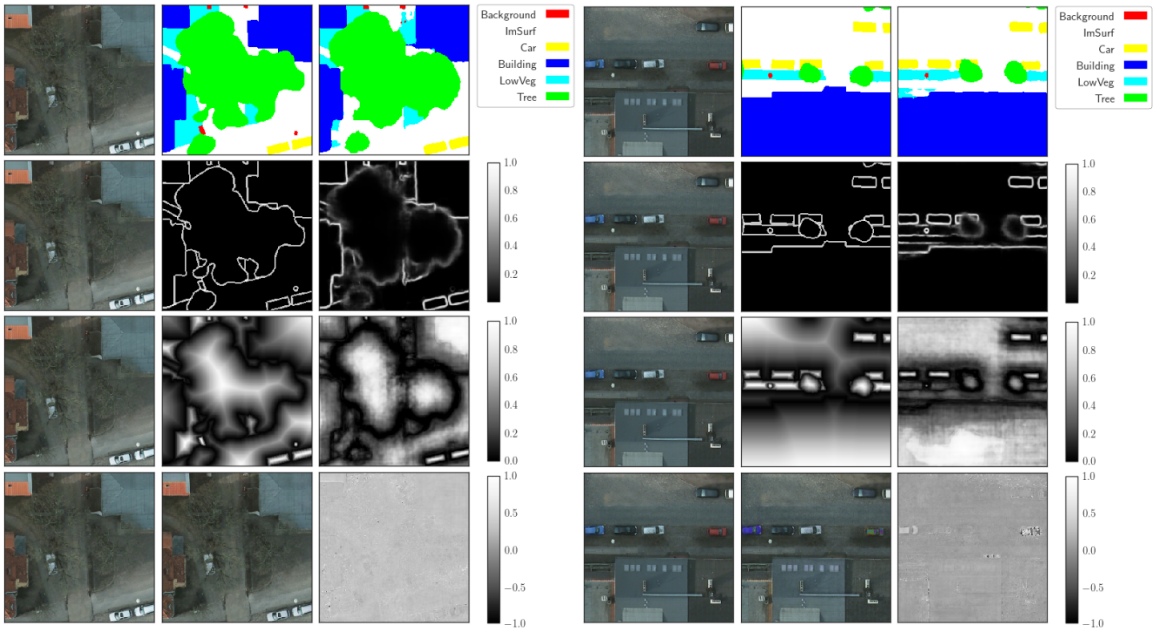


Figura 25: Exemplo do resultado de cadas uma das tarefas do aprendizado multitarefa da ResUneta. No topo o resultado da tarefa principal de segmentação semântica. A segunda, de cima para baixo, o resultado da detecção de bordas. A terceira é a transformação distância. E a quarta e última linha, é a transformação em HSV já reconstruída de volta em RGB (segunda coluna da esquerda para direita) e a diferença entre as duas imagens em RGB (terceira coluna da esquerda para direita.

Para inserir na arquitetura o aprendizado multitarefa foi escolhido um arranjo condicionada diferentemente das abordagens tradicionais, que utilizam um treinamento independente entre as tarefas. Essa mudança, em comparação a um aprendizado multitarefa independente gerou uma maior estabilidade e velocidade na convergência do treinamento.

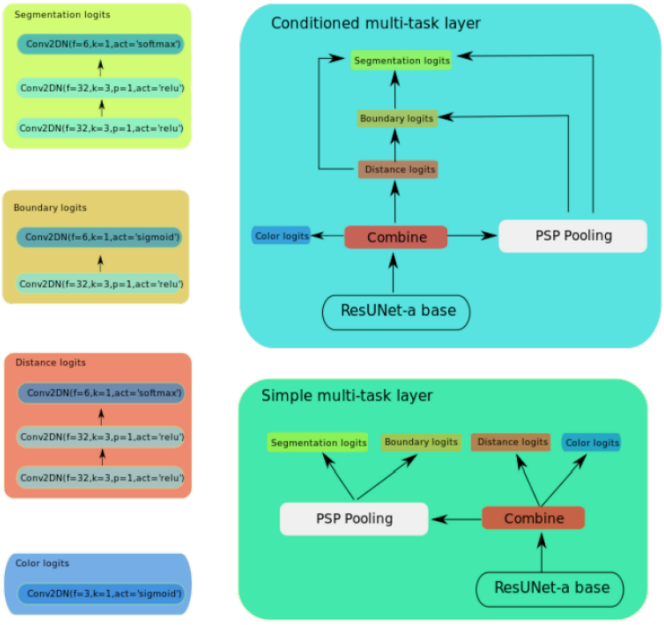


Figura 26: Arquitetura do aprendizado multitarefa na ResUnet-a

3 Metodologia

a Preparação dos dados

Para performar a tarefa de detecção de mudanças em zonas de desmatamento e não desmatamento, foram utilizadas duas imagens, obtidas com o satélite Landsat 8, sendo elas ocorridas entre Julho de 2018 e Julho 2019. A área usada nesse estudo é localizada na região do Estado do Pará no Brasil, centrada nas coordenadas 06°54' 16" S e 055°11' 52" O, o qual possui 152.475,00 km^2 desmatados, cerca de 34% da sua região total, que é a maior taxa de desmatamento do Brasil. Os dados foram obtidos na base de dados do PRODES a qual é aberta e pode ser obtida no seguinte endereço: `http://terrabrasilis.dpi.inpe.br/map/deforestation`. A evolução da taxa de desmatamento pode ser observada na figura 27.

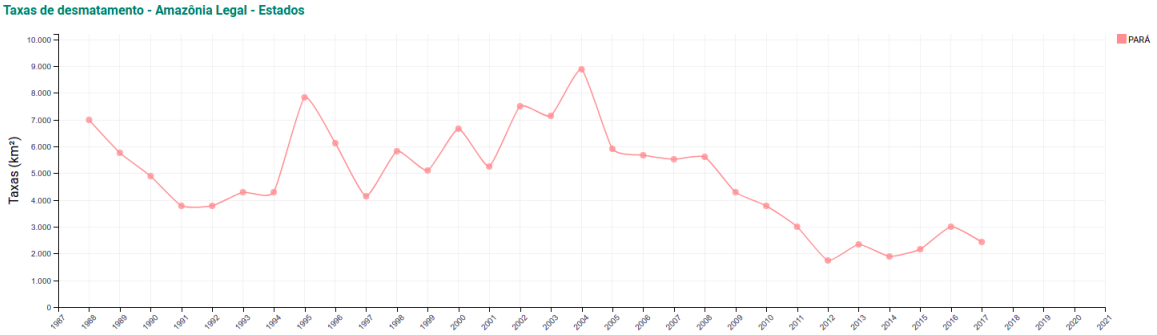


Figura 27: Taxa de desmatamento no Estado do Pará, Brasil

Ambas as imagens possuem resoluções de 5905x3064, porém foram recortadas nas resoluções 5900x3060 para prevenir perdas de *patches* durante a extração e tornar a divisão em retalhos (*tiles*) mais uniforme. Das imagens obtidas pelo Landsat 8, foi utilizado somente 7 bandas: Coastal/Aerosol, Azul, Verde, Vermelho, NIR, SWIR-1, and SWIR-2. Para usá-las em uma abordagem de change detection nós adotamos um método denominado *early fusion*, inspirado nos artigos [11], [8], o qual concatena as duas imagens antes de entrar na rede neural transformando as duas imagens de 7 bandas em um única de 14 bandas.

Além das imagens de entrada, utilizamos 3 classes:

- 0: Sem desmatamento
- 1: Desmatamento atual
- 2: Desmatamento passado

Similarmente a [11] nós talhamos a imagem em 15 [Figura 28] para generalizar o aprendizado. Nós separamos 60% das 15 partes para o treino, 13% para validação e 27% para teste. Os talhos 3, 2, 7, 10, 9, 1, 14, 4, 11 foram usados na etapa de treino, 15, 12 na validação e 5, 13, 8, 6 no teste.

Para treinar o modelo com imagens de satélite é necessário picotar a imagem em vários recortes menores, visto que a imagem de satélite possui uma resolução muito alta e seria necessário um computador com muita memória nas placas de vídeo para conseguir processar ela por inteira. Sendo assim, foi extraído recortes pequenos da imagem original com um tamanho adequado para ser alocado nas placa de video e assim ser processadas pela rede neural. Dentre esses recortes extraídos da imagem, selecionamos apenas os quais possuem 2% de desmatamento atual com o objetivo de lidar com o problema de desbalanceamento da base de dados, visto que possui muito mais ocorrências da classe não desmatamento e assim teríamos recortes contendo apenas ocorrências dessa classe o que poderia gerar uma má generalização no treinamento. Entretanto, filtrando recortes com apenas essa quantidade de desmatamento diminui consideravelmente a quantidade de dados à ser utilizado no treinamento. Por causa dessa quantidade escassa de dados, foi escolhido recortes um

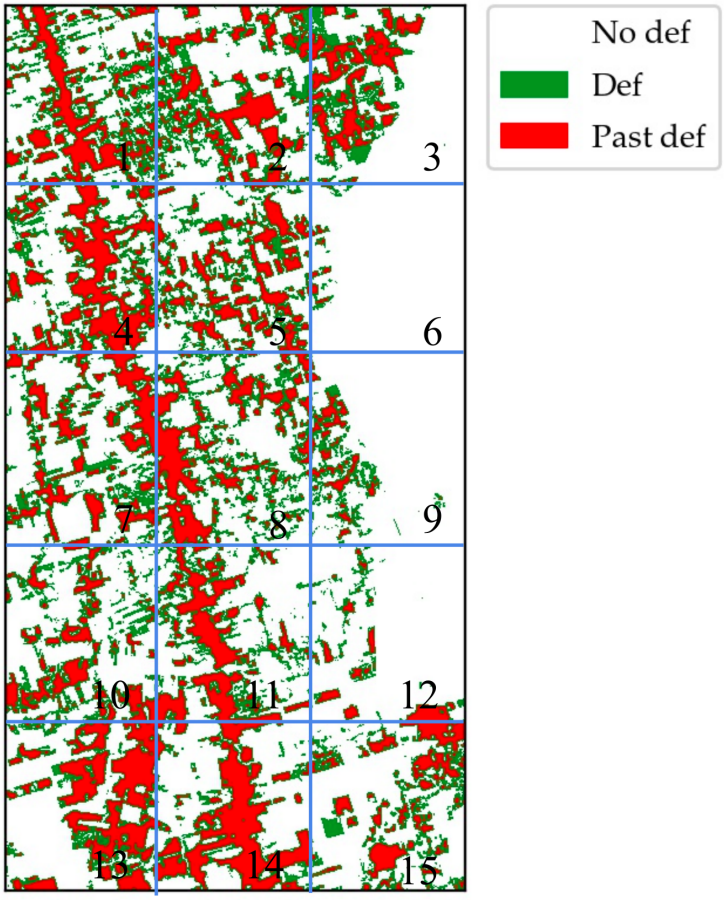


Figura 28: Área de estudo dividido em 15 talhos

pouco menores de 128x128 e uma certa sobreposição entre eles com um espaçamento (*stride*) de 32 entre um talho e seu subsequente, visando aumentar o conjunto de treino. Na etapa de teste os recortes continuam sendo de tamanho 128x128, porém utilizamos uma janela de 128, ou seja, sem sobreposição entre os recortes. Seguindo esse pré-processamento na etapa de treino obtemos uma melhor distribuição das classes que pode ser observada na figura 29.

Além dessas estratégias na etapa de treino para melhorar a distribuição das classes, a quantidade de imagens continuava pequena e portanto adotamos uma outra estratégia de sobre-amostragem durante o treino. Além dos recortes de tamanho 128x128 extraídos, utilizamos uma espécie de *data augmentation* nas imagens: rotação anti-horária de 90º e 180º e espelhamento horizontal. Com isso multiplicamos o número total de recortes extraídos por 4, aumentando expressivamente o tamanho dos dados utilizado no treinamento do modelo.

Além das referências para a tarefa principal de segmentação semântica, outras três referências foram usadas para as tarefas de detecção de bordas, transformação distância e o CVA. As referências das duas primeiras tarefas foram geradas por algoritmos padrões da biblioteca OpenCV [21] a partir das referências de segmentação semântica, de acordo com o artigo original da ResUnet-a [14]. Já as referências utilizadas pelo algoritmo do CVA foram calculadas diretamente das imagens de satélite de entrada, diferentemente das outras rotinas. Na seção 3.c será explicado com mais detalhes como essas referências foram geradas.

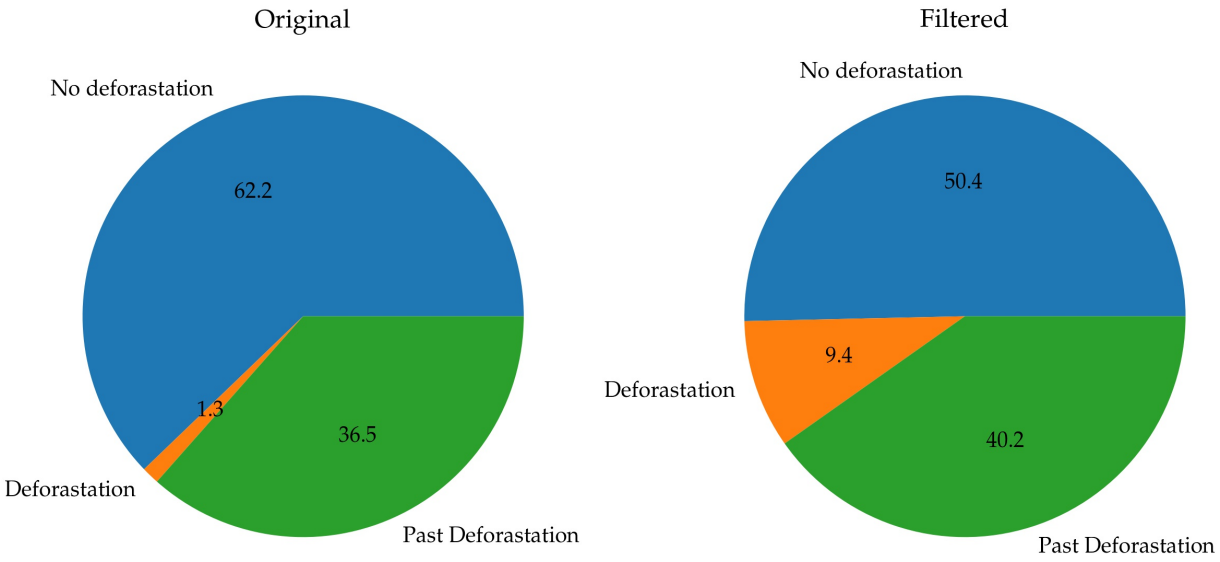


Figura 29: Ocorrências das classes

b Arquitetura da rede neural utilizada

Como dito anteriormente na seção d do capítulo 2, a rede utilizada foi a ResUnet-a, a qual utiliza um multitasking condicionado para aumentar a performance da tarefa principal de segmentação semântica [Figura 25]. Diferentemente da implementação original, foi removido a tarefa de transformação HSV, pois ela não fazia sentido no âmbito problema de *change detection*, visto que é utilizado duas imagens de entrada concatenadas com 7 bandas cada. Alguns experimentos foram feitos tentando se utilizar da transformação das imagens em HSV das duas imagens, pegando somente os canais RGB e tentando predizer na saída desta tarefa uma imagem de 6 canais, 3 canais HSV para cada imagem T1 e T2. Entretanto, devido a problemas de convergência com esta tarefa foi decidido remover a tarefa.

Inspirado pela abordagem multitarefa, nós decidimos adicionar mais uma tarefa, o CVA, a qual focaria nas mudanças ocorridas entre as imagens T1 e T2, ou seja, a classe desmatamento atual. No nosso método, todas as tarefas se utilizam das 3 classes originais exceto a tarefa de CVA, já que se trata de uma classificação binária. Ele classifica os pixels em mudanças (desmatamento atual) e não mudanças. A ideia de usar a magnitude do CVA para gerar as referências do aprendizado supervisionado como mais uma tarefa de toda arquitetura da rede é estimular as etapas de codificação e decodificação a aprender as informações geradas pela técnica do CVA e com isso aprimorar a segmentação dos pixels da tarefa principal.

Assim, a arquitetura final do nosso método fica com 4 tarefas a serem aprendidas no treinamento: segmentação semântica, transformação distância, detecção de bordas e *change vector analysis*. O esquema de todas as tarefas podem ser vistas na figura

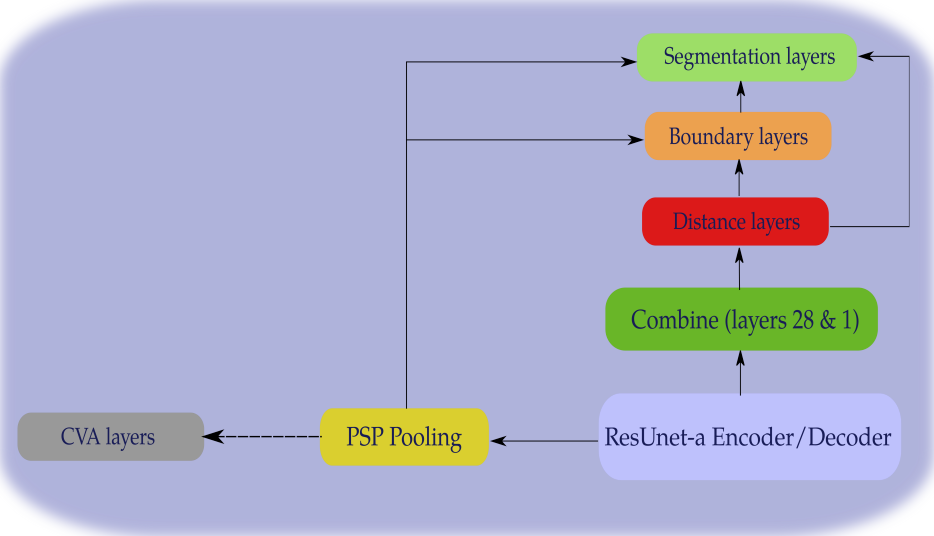


Figura 30: ResUnet-a e todas as tarefas aprendidas

A camada do CVA foi posta ao final da camada de *PSP Pooling* fora da estrutura condicionada pois foi o lugar onde obtivemos os melhores resultados. Outras tentativas foram feitas, como por exemplo, concatenar a saída da camada de CVA na entrada de todas as outras tarefas ou somente na camada de segmentação semântica, porém os resultados foram todos piores que o aprendizado sozinho da camada de CVA, evidenciando que a a contribuição não auxiliou tanto o aprendizado das outras tarefas.

As camadas utilizadas em cada tarefa podem ser vistas nas tabelas 1, 2, 3, 4.

Layer #	Layer type
input	Concat(Bound, Dist, PSP pooling)
1	Conv(f=32, k=3, p=1)
2	BatchNorm
3	ReLu
4	Conv(f=32, k=3, p=1)
6	BatchNorm
7	ReLu
8	Conv(f=3, k=1, p=0)
9	Softmax

Tabela 1: Segmentation head layers

Layer #	Layer type
input	Concat(Dist, PSP pooling)
1	Conv(f=32, k=3, p=1)
2	BatchNorm
3	ReLu
8	Conv(f=3, k=1, p=0)
9	Sigmoid

Tabela 2: Boundary head layers

Layer #	Layer type
input	Combine(Layer 28, Layer 1)
1	Conv(f=32, k=3, p=1)
2	BatchNorm
3	ReLu
4	Conv(f=32, k=3, p=1)
6	BatchNorm
7	ReLu
8	Conv(f=3, k=1, p=0)
9	Softmax

Tabela 3: Distance transform head layers

Layer #	Layer type
input	PSP pooling
1	Conv(f=32, k=3, p=1)
2	BatchNorm
3	ReLu
4	Conv(f=32, k=3, p=1)
6	BatchNorm
7	ReLu
8	Conv(f=2 k=1, p=0)
9	Softmax

Tabela 4: CVA head layers

c Change Vector Analysis

Change vector analysis, também conhecido como CVA, é um algoritmo muito utilizado em problemas de change detection, o qual cada pixel possui uma magnitude e uma fase. A magnitude representa a intensidade da mudança e a fase o tipo de mudança, seja ela de vegetação, urbana, etc ... Neste trabalho usaremos exclusivamente a magnitude. Para determinar zonas de mudanças de não mudanças um valor de limiar deve ser escolhido. Visando detectar desmatamento seguimos a linha do trabalhos [18], [11] e selecionamos *Normalized Difference Vegetation Index* (NDVI) e o *Bare Soil Index* (BI) para calcular o CVA. O NDVI quantifica vegetação, mensurando o nível de clorofila nas folhas através das bandas *near-infrared* (NIR) e vermelha. O BI por sua vez, é calculado para distinguir terras de agricultura e não-agricultura. Combinando ambos os índices, podemos especificar nossa detecção em apenas vegetação.

Essas operações foram performadas para cada pixel nas datas T1 e T2. O índices NDVI e BI podem ser calculados com as seguintes equações 10 e 11:

$$NDVI = \frac{NIR - RED}{NIR + RED} \tag{10}$$

$$BI = \frac{(SWIR + RED) - (SWIR + BLUE)}{(SWIR - RED) - (SWIR - BLUE)} \tag{11}$$

Onde, *NIR*, *RED*, *SWIR* and *BLUE*, são mensuramentos da refletância spectral adquiridos nas bandas infravermelho próximo, vermelho, infravermelho de onda curta e azul.

A vetor magnitude é representado por S na equação (12).

$$S = \sqrt{(NDVI_2 - NDVI_1)^2 + (BI_2 - BI_1)^2} \tag{12}$$

Sendo os valores dos pixels de *SWIR* e *NIR* possuem valores muito altos, nós decidimos normalizar T1 e T2 antes dos cálculos. fazendo isso, a magnitude S estaria num intervalo pequeno e seria mais fácil de encontrar o limiar. O limiar será usado para transformar a imagem S em uma imagem binária $S^{'}$ [(13)]. Sendo assim, podemos tratar a tarefa do CVA como uma simples classificação binária.

$$S^{'}(x_{i,j}) = \begin{cases} 1, & \text{se } x_{i,j} > \text{limiar} \\ 0, & \text{caso contrário} \end{cases} \tag{13}$$

Onde $x_{i,j}$ é o pixel de S na posição i, j.

Para achar o melhor limiar nós calculamos uma curva ROC [Figura 31] com S e somente a referência de desmatamento atual.

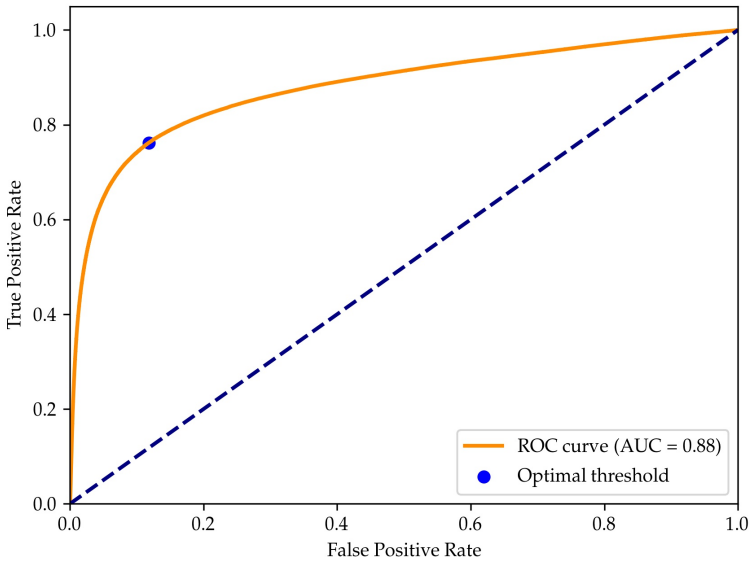
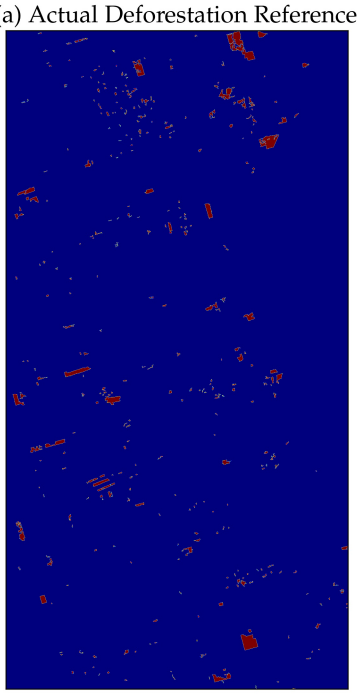


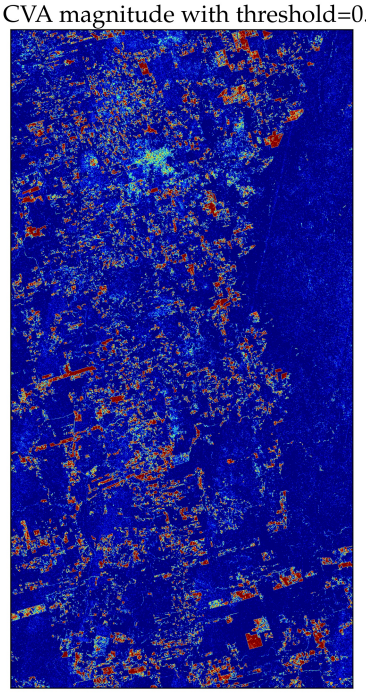
Figura 31: CVA ROC curve

Fazendo a curva ROC obtemos um limiar ótimo de 0.34, todavia também fizemos testes com valores de 0.5 e 1.0 para estudar a influência do ruído presente no CVA na rede neural, sabendo que para limiares mais baixos possuímos mais ruído na magnitude do CVA comparado à referência de desmatamento atual somente [Figura 32].

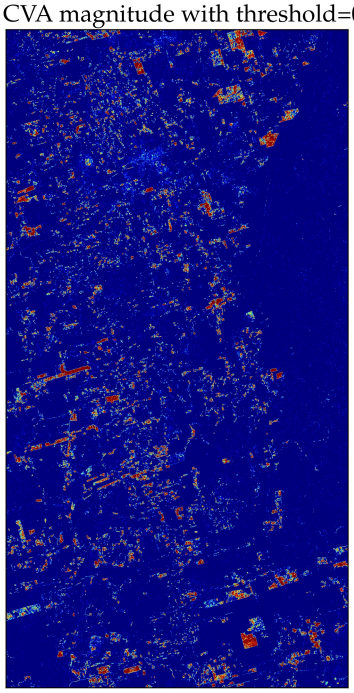
(a) Actual Deforestation Reference



(b) CVA magnitude with threshold=0.34



(c) CVA magnitude with threshold=0.5



(d) CVA magnitude with threshold=1.0

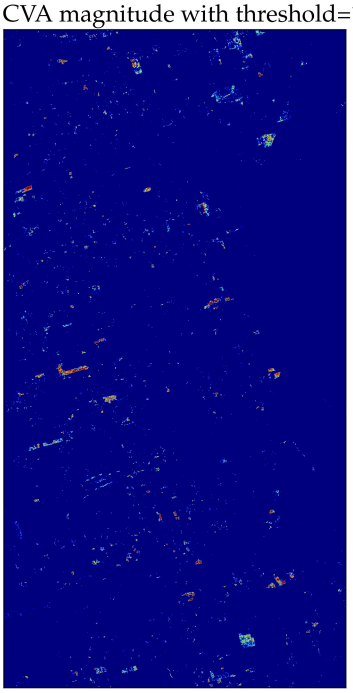


Figura 32: CVA magnitudes compared with Actual Deforestation Reference

d Funções de custo

Um grande problema de se trabalhar com o dataset escolhido era o grande desbalanceamento das classes. Durante o pré-processamento algumas abordagens foram tomadas para mitigar esse problema, assim como mostrada na seção 3.a. Entretanto, mesmo atenuando o problema nessa etapa anterior ao treino, ainda é necessário utilizar outros métodos para que a rede aprenda as características corretas de todas as classes e não somente da classe com mais ocorrências, ou seja, *não desmatamento*. Visto isso, a solução seria balancear as classes na própria função de custo. Na

implementação original da ResUnet-a [14] foi utilizada a *Dice loss* com complemento como função de custo com o objetivo de convergir melhor o treino do aprendizado multitarefa como um todo. Entretanto, para o nosso problema com a tarefa adicional do CVA ela não performou tão bem quanto o esperado e, portanto, decidimos utilizar outras funções de custo. Sendo assim, no lugar da Dice loss, utilizamos a *Binary Cross Entropy* (BCE) [Equação 15] para a tarefa de detecção de bordas, pois se trata de uma classificação multi-label ou multi-referência. Isso se dá oriundo do fato de que as bordas de uma imagem podem ser as mesmas para classes diferentes, como pode ser visto na figura 33 para as classes de *não desmatamento* (No def) e *desmatamento passado* (Past def).

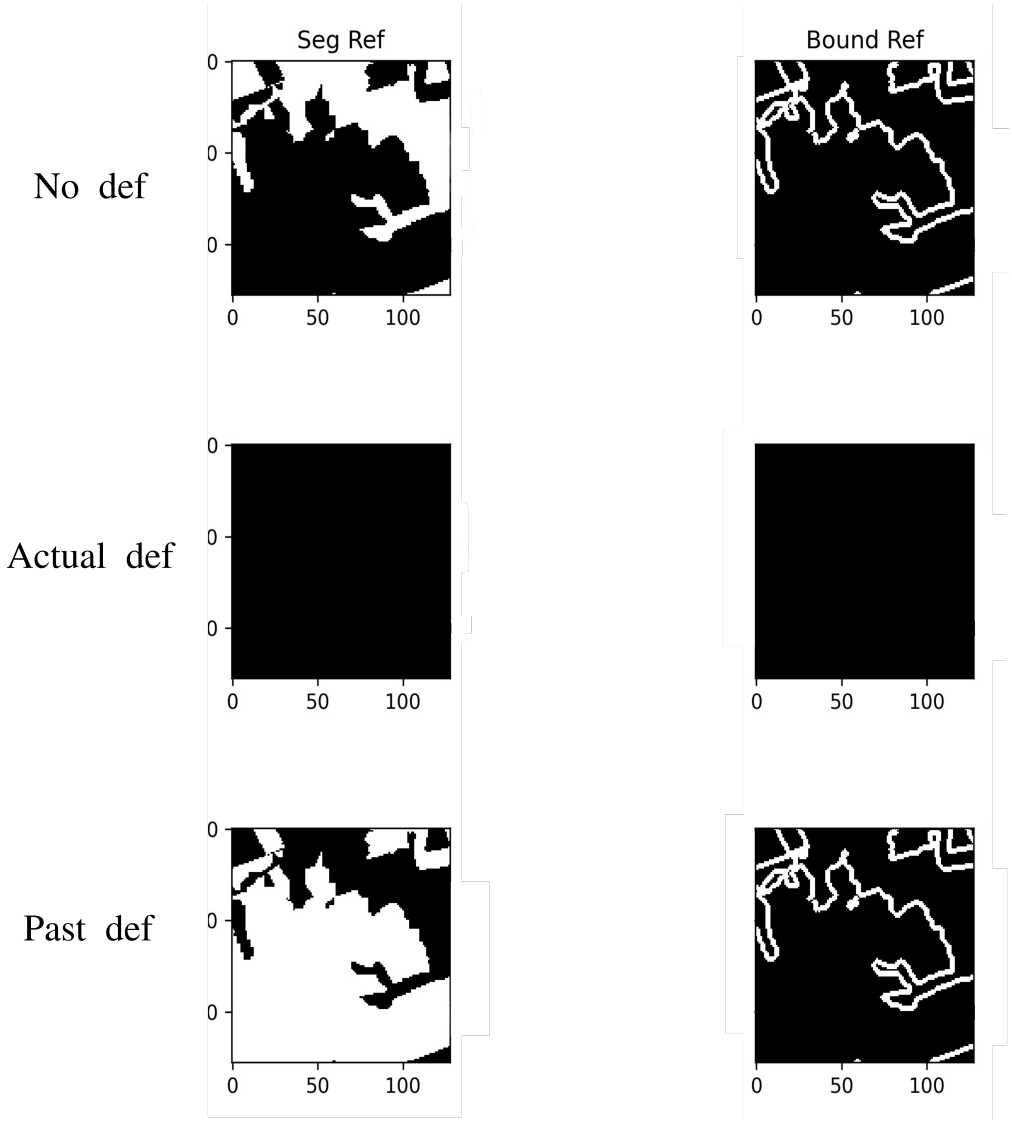


Figura 33: Comparação das referências entre detecção de bordas e segmentação semântica

Já para a tarefa da transformação distância foi utilizado a *Mean Squared Error* (MSE) [Equação 14] visto que a tarefa se trata de uma regressão. Para a tarefa do CVA foi utilizado a clássica *Cross Entropy* (CE) [Equação 16], pois trata-se de uma classificação multi classe. Finalmente, para a tarefa principal de segmentação semântica foi utilizado a versão ponderada da entropia cruzada, *Weighted Cross Entropy* (WCE) [Equação 17] para lidar com o desbalanceamento das classes.

$$MSE(p,q) = \frac{1}{N}\sum_{i}^{N}(p(x) - q(x))^2 \tag{14}$$

$$BCE(p,q) = -\frac{1}{N}\sum_{i}^{N} p(x)log(q(x)) + (1-p(x))log((1-q(x))) \tag{15}$$

$$CE(p,q) = -\frac{1}{N}\sum_{i}^{N} p(x)log(q(x)) \tag{16}$$

$$WCE(p,q) = -\frac{1}{N}\sum_{i}^{N} Wp(x)log(q(x)) \tag{17}$$

Sendo N o número de amostras, x o vetor de entrada (no caso o patch com 14 bandas) $q(x)$ a probabilidade de cada classe obtida pelo modelo e $p(x)$ é a probabilidade de cada classe no dataset P.

Mesmo balanceando cada classe com a função de custo WCE ainda era difícil para a rede aprender as características da classe **desmatamento atual**. No geral, a rede tendia a errar a classe de desmatamento atual como desmatamento passado, já que a última possui muito mais ocorrências que a primeira. Visto esse problema, decidimos penalizar a classe de desmatamento passado, obrigando a rede a não aprender mais essa classe. A consequência dessa abordagem faz com que ela classifique os pixels associados a desmatamento passado como alguma das outras duas classes: desmatamento atual ou não desmatamento. Contudo, a tendência a esse erro se torna menor para a classe desmatamento atual já que ela possui menos ocorrências que a classe de não desmatamento. Sendo assim, ao classificar erradamente os pixels da classe desmatamento passado o modelo tenderá a dar probabilidades maiores para **não desmatamento**. Como o objetivo é classificar os pixels da classe desmatamento atual, não tem muita importância em errar a classe desmatamento passado, juntando com a de não desmatamento.

Sendo assim, escolhemos os pesos $W = [0.4, 0.6, 0]$ para a função de custo WCE com o objetivo de penalizar a classe de desmatamento passado e ligeiramente melhorar a classificação para a classe desmatamento atual. Outros valores foram escolhidos, porém os resultados se mostraram pior conforme a discrepância entre as classes *não desmatamento* e *desmatamento atual* aumentava. Outras maneiras de calcular pesos foram efetuadas como por exemplo utilizando a relação $W_i = Pixels\ Totais/Pixels\ da\ classe$, porém os resultados não foram muito promissores e optamos por utilizá-los normalizados.

A função de custo total [Equação 18] utilizada foi a soma das funções de custo descritas anteriormente. Os pesos de todas as funções de custo utilizadas foi de 1.0.

$$Loss = WCE_{Seg} + CE_{CVA} + BCE_{Bound} + MSE_{Dist} \tag{18}$$

4 Resultados

Para avaliar os resultados fizemos uma curva de Precision-Recall baseado na tarefa principal de segmentação semântica. Nós selecionamos a probabilidade da classe **desmatamento atual** para efetuar a curva e após sua finalização foi calculado o *Mean Average Precision* (mAP) dela. A ideia é avaliar se a nova tarefa do CVA contribuiu para melhorar a classificação da classe **desmatamento atual**. Além de avaliar se a nova tarefa proposta nesse trabalho realmente melhora o aprendizado para classificar a classe de desmatamento atual, também resolvemos avaliar a contribuição de cada uma das outras tarefas em relação a esse mesmo problema. Para estudar as contribuições, foram feitos 4 treinamentos. Primeiro treinamos um modelo base assim como descrito na seção 3.b, utilizando somente as tarefas oriundas da implementação original [14], porém sem a tarefa de transformação em HSV. Em seguida, treinamos esse mesmo modelo base com a adição do CVA, que é a proposta desse trabalho. Após esse treino, realizamos outros 2 treinamentos retirando as tarefas de detecção de bordas e a transformação distância da arquitetura da rede. As curvas de Precision-Recall e seus mAP de cada um desses treinamentos podem ser observados na figura 34.

Todos os treinos foram feitos utilizando *learning rate* de 1×10^{-3}, *batch size* 32 e *weight decay* de 0.01 e utilizamos o otimizador SGD, o qual apresentou resultados muito melhores que o Adam. Também foi utilizado *early stopping* em todos os treinamentos com uma paciência de 10 épocas sem a função de custo melhorar. Sendo assim, os modelos **Baseline**, **Baseline+CVA**, **Baseline+CVA-Dist**, **Baseline+CVA-Bound** duraram 70, 58 , 48, 52 épocas, respectivamente.

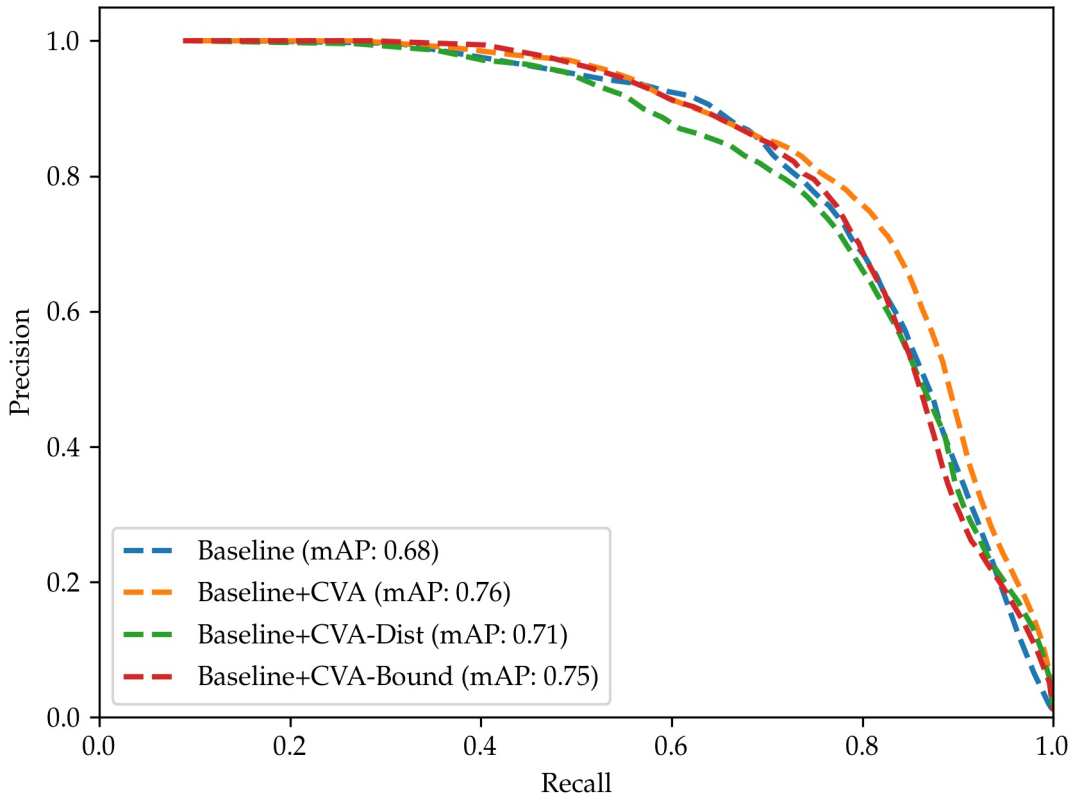


Figura 34: Curvas de Precision-Recall e seus mAP comparando a contribuição de cada uma das tarefas.

Como podemos ver que o método com as tarefas originais da ResUnet-a juntamente com a tarefa de CVA foi o melhor método (curva laranja figura 34), tendo um mAP de 0.76 e um ganho de 8% no comparado com a Baseline. Removendo a transformação distância do modelo proposto (Baseline + CVA) podemos observar um mAP de 0.71 e, portanto, uma perde de 5%. Aplicando esse mesmo procedimento para a tarefa de detecção de bordas, obtivemos um mAP de 0.75 e uma perda de 1%,

mostrando que essa foi a tarefa que menos contribuiu para o aprendizado da rede neural. Sendo assim, podemos concluir que a tarefa que mais contribuiu para a classificação dos pixels da classe desmatamento atual foi a tarefa do CVA.

Outra observação interessante é avaliar o resultado da segmentação de **desmatamento atual** sobre alguns recortes inferidos. Nas figuras 35 e 36 podemos observar que a área sem desmatamento possui alguns pontos de probabilidades altas para o modelo base na figura 36, enquanto no exato mesmo modelo treinado com a tarefa de CVA, houve uma certa diminuição dessas probabilidades, ou seja, uma maior confiança de que esses pixels não pertencem a classe desmatamento atual.

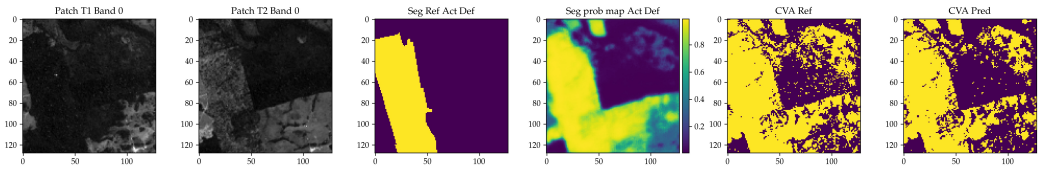


Figura 35: Inferência sobre um recorte do conjunto de teste feito pelo modelo base com adição da tarefa do CVA. Da esquerda para direita, as duas primeiras imagens são a primeira banda das imagens T1 e T2 de entrada, seguido pela referência e predição dos mapas de probabilidade da classe desmatamento atual da tarefa principal de segmentação semântica e terminando as duas últimas com as referências geradas pelo CVA e sua predição gerada pela tarefa adicionada

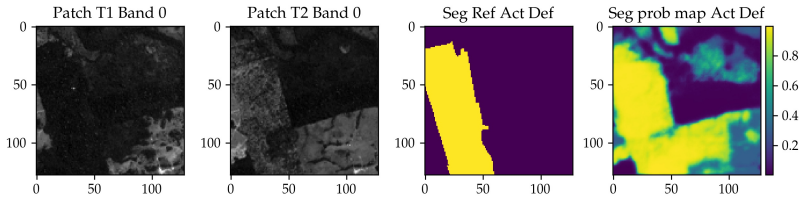


Figura 36: Inferência sobre um recorte do conjunto de teste feito pelo modelo base com adição da tarefa do CVA. Da esquerda para direita, as duas primeiras imagens são a primeira banda das imagens T1 e T2 de entrada, seguido pela referência e predição dos mapas de probabilidade da classe desmatamento atual da tarefa principal de segmentação semântica

Nas figuras 37 e 38 podemos ver na referência utilizada na terefa do CVA na figura 37 que possui algumas interseções com as referências utilizadas para a segmentação de desmatamento atual. Se compararmos essas interseções com os mapas de probabilidades gerados pelos modelo base e com a adição da tarefa de CVA, podemos ver que os mapas possuem zonas de probabilidades são maiores nessas regiões para o modelo com adição do CVA, o que sugere que a adição da supervisão da magnitude do CVA fez com que o modelo tivesse mais confiança em atribuir o rótulo de desmatamento atual aos pixels dessas regiões.

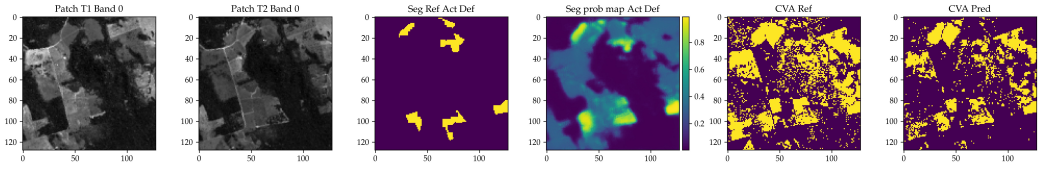


Figura 37: Inferência sobre um recorte do conjunto de teste feito pelo modelo base com adição da tarefa do CVA. Da esquerda para direita, as duas primeiras imagens são a primeira banda das imagens T1 e T2 de entrada, seguido pela referência e predição dos mapas de probabilidade da classe desmatamento atual da tarefa principal de segmentação semântica e terminando as duas últimas com as referências geradas pelo CVA e sua predição gerada pela tarefa adicionada

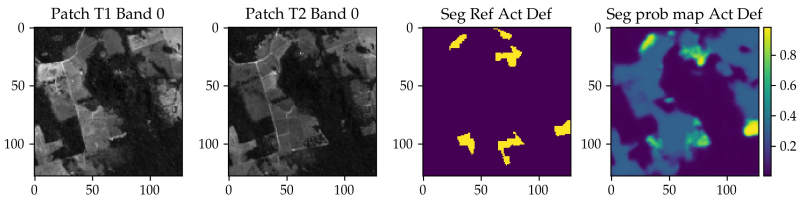


Figura 38: Inferência sobre um recorte do conjunto de teste feito pelo modelo base com adição da tarefa do CVA. Da esquerda para direita, as duas primeiras imagens são a primeira banda das imagens T1 e T2 de entrada, seguido pela referência e predição dos mapas de probabilidade da classe desmatamento atual da tarefa principal de segmentação semântica

Além das contribuições de cada tarefa, também foi estudado a influência no *mAP* [Figura 39] do limiar utilizado na referência criada com o CVA. Para isso, foi feito três treinos utilizando com a ResUnet-a, com suas devidas modificações, e a adição do CVA. Em cada um dos treinos modificamos a referência gerada pelo CVA mudando o limiar. Nós descobrimos que o limiar sugerido pela curva ROC [Figura 31] foi realmente a melhot escolha, mesmo contendo mais ruído que o esperado, comparado com a referência de desmatamento atual somente na figura 32. Ademais, nós tivemos uma variação de 4% no máximo, o que indica que o treinamento não foi tão suscetível a variações do limiar como abordagens tradicionais de detecção de mudanças utilizando o algoritmo CVA [6], [7], as quais são mais dependentes da escolha do limiar.

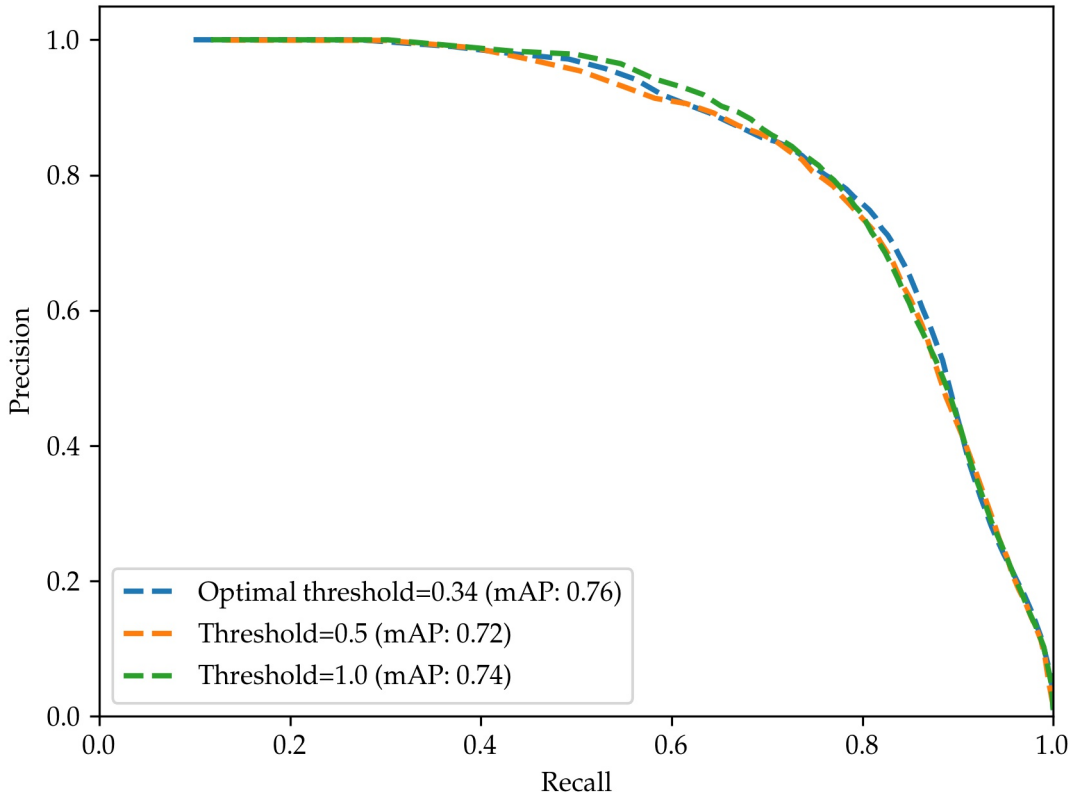


Figura 39: Curvas de Precision-Recall e seus mAP comparando os limiares utilizados nas referências do CVA.

Outro estudo interessante, foi quanto ao número de parâmetros da rede neural utilizada. Devido a escassez de dados utilizados no treinamento, resolvemos verificar se a rede tinha chegado no limite do aprendizado dela. Como não temos mais dados, fizemos o teste com uma versão menor da ResUnet-a. Desta forma, foi removido todas as convoluções com taxas de dilatação diferentes de 1 das etapas de codificação e decodificação. Com isso, obtemos uma redução de $\sim$ **57MM** to $\sim$ **40MM**

e um ganho de 3% no *mAP* [Figura 40], o que mostra e o modelo original poderia ter performado melhor com uma maior quantidade de dados.

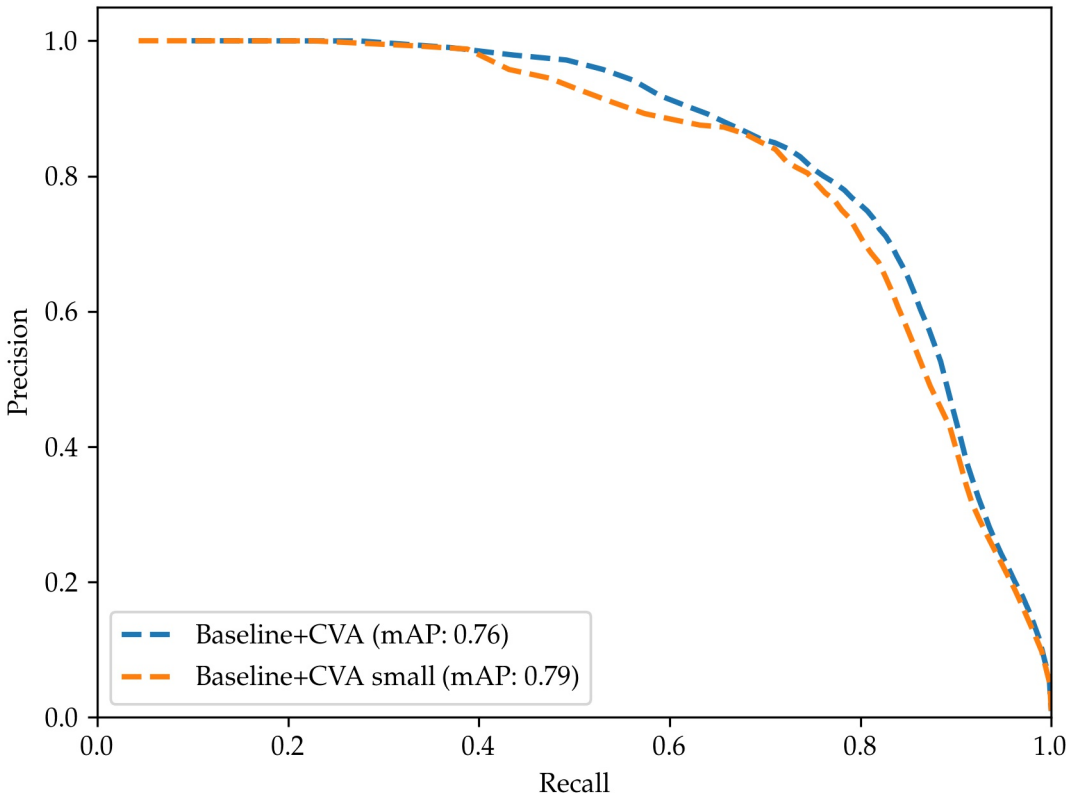


Figura 40: Curvas de Precision-Recall da ResUnet-a com CVA e sua versão menor sem convoluções dilatadas extras

5 Conclusão

Nós apresentamos um novo método para ser usado em problemas de detecção de mudanças (*change detection*) para zonas de vegetação utilizando segmentação semântica. A maior qualidade desse método se deve ao fato de sua implementação ser muito simples, pois somente é preciso adicionar mais uma tarefa à rede neural principal. Além disso, não é preciso anotar nenhuma referência para esta tarefa, já que elas são geradas pelo algoritmo *Change Vector Analysis* diretamente das imagens de entrada, diferentemente das outras tarefas propostas por [14] as quais utilizam as próprias referências da segmentação semântica. A nova tarefa é composta de apenas duas camadas convolucionais normalizadas (camada convolucional + batch normalization) o que torna o aumento de parâmetros da rede neural quase nulo. Por fim, também não houve problemas de convergência durante o aprendizado multitarefa envolvendo a tarefa da magnitude do CVA e pudemos treinar a tarefa com a contribuição de 1.0. Sendo assim, devido aos resultados comparados com o modelo base podemos dizer que a adição de mais uma supervisão com a magnitude do CVA pode ser uma boa abordagem para problemas de detecção de mudanças em zonas de vegetação.

Além da magnitude do CVA, este trabalho abre caminho para futuras pesquisas como a utilização de mais uma supervisão utilizando a fase do algoritmo do CVA e o cálculo da magnitude utilizando todas as bandas das imagens de entrada ao invés dos índices escolhidos.

Referências

[1] H. Escobar, “Jornal usp: Desmatamento da amazônia dispara de novo em 2020.” [Online]. Available: https://jornal.usp.br/ciencias/desmatamento-da-amazonia-dispara-de-novo-em-2020/

[2] T. E. o. E. Britannica, “Amazon Rainforest,” https://www.britannica.com/place/Amazon-Rainforest, 2019.

[3] A. Thomson, “Biodiversity and the Amazon Rainforest,” https://www.greenpeace.org/usa/biodiversity-and-the-amazon-rainforest/: :text=the2020.

[4] R. Butler, “Saving What Remains, MEDICINAL PLANTS,” https://rainforests.mongabay.com/1007.htm, 2012.

[5] J. Holland, “Nature’s pharmacy: The remarkable plants of the Amazon rainforest and what they may cure,” https://www.telegraph.co.uk/travel/cruises/articles/how-to-be-a-botanical-buff/, 2019.

[6] D. Lu, P. Mausel, E. Brondízio, and E. Moran, “Change detection techniques,” *International Journal of Remote Sensing*, vol. 25, no. 12, pp. 2365–2401, 2004.

[7] S. Mishra, P. Shrivastava, and P. Dhurvey, “Change Detection Techniques in Remote Sensing: A Review,” *International Journal of Wireless and Mobile Communication for Industrial Systems*, vol. 4, no. 1, pp. 1–8, 2017.

[8] R. C. Daudt, B. Le Saux, A. Boulch, and Y. Gousseau, “Urban change detection for multispectral earth observation using convolutional neural networks,” *International Geoscience and Remote Sensing Symposium (IGARSS)*, vol. 2018-July, pp. 2115–2118, 2018.

[9] Y. Zhan, K. Fu, M. Yan, X. Sun, H. Wang, and X. Qiu, “Change detection based on deep siamese convolutional network for optical aerial images,” *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 10, pp. 1845–1849, 2017.

[10] S. Zagoruyko and N. Komodakis, “Learning to compare image patches via convolutional neural networks,” 2015.

[11] M. Ortega Adarme, R. Queiroz Feitosa, P. Nigri Happ, C. Aparecido De Almeida, and A. Rodrigues Gomes, “Evaluation of deep learning techniques for deforestation detection in the brazilian amazon and cerrado biomes from remote sensing imagery,” *Remote Sensing*, vol. 12, no. 6, p. 910, 2020.

[12] S. Rakshit, S. Debnath, and D. Mondal, “Identifying land patterns from satellite imagery in amazon rainforest using deep learning,” *CoRR*, vol. abs/1809.00340, 2018. [Online]. Available: http://arxiv.org/abs/1809.00340

[13] Z. Zhang, G. Vosselman, M. Gerke, D. Tuia, and M. Y. Yang, “Change detection between multimodal remote sensing data using siamese CNN,” *CoRR*, vol. abs/1807.09562, 2018. [Online]. Available: http://arxiv.org/abs/1807.09562

[14] F. I. Diakogiannis, F. Waldner, P. Caccetta, and C. Wu, “Resunet-a: A deep learning framework for semantic segmentation of remotely sensed data,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 162, pp. 94 – 114, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924271620300149

[15] Z. Zhang, Q. Liu, and Y. Wang, “Road extraction by deep residual u-net,” *CoRR*, vol. abs/1711.10684, 2017. [Online]. Available: http://arxiv.org/abs/1711.10684

[16] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[18] S. tong si, L. Pham, and V. PHAM, “Land cover change analysis using change vector analysis method in duy tien district, ha nam province in vietnam,” 10 2009.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[20] R. Zadeh, “Distributed Algorithms and Optimization,” http://stanford.edu/ rezab/classes/cme323/S15/notes/lec11.pdf, 2015.

[21] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.