MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

EDUARDO CESAR NOGUEIRA COUTINHO
JOÃO MARCOS GRIS
1º Ten VICTOR FEITOSA DE CARVALHO SOUZA

DEEP LEARNING NA ANÁLISE DE IMAGENS PARA DETECÇÃO DE REGIÕES DE DESMATAMENTO

Rio de Janeiro
2017

INSTITUTO MILITAR DE ENGENHARIA

EDUARDO CESAR NOGUEIRA COUTINHO
JOÃO MARCOS GRIS
1º Ten VICTOR FEITOSA DE CARVALHO SOUZA

DEEP LEARNING NA ANÁLISE DE IMAGENS PARA DETECÇÃO DE REGIÕES DE DESMATAMENTO

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Prof. Ronaldo Ribeiro Goldschmidt - D.Sc.
Co-Orientador: Maj Vitor Henrique Pereira Draeger - M.Sc.

Rio de Janeiro
2017

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 - Praia Vermelha
Rio de Janeiro - RJ CEP 22290-270

526.982 Coutinho, Eduardo Cesar Nogueira
C871d Deep Learning na Análise de Imagens para Detecção de Regiões de Desmatamento / Eduardo Cesar Nogueira Coutinho; João Marcos Gris; Victor Feitosa de Carvalho Souza; orientados por Ronaldo Ribeiro Goldschmidt ; Vitor Henrique Pereira Draeger - Rio de Janeiro: Instituto Militar de Engenharia, 2017.

62p.: il.

Projeto de Fim de Curso (PROFIC) - Instituto Militar de Engenharia, Rio de Janeiro, 2017.

1. Curso de Engenharia de Computação - Projeto de Fim de Curso. 2. Inteligência Artificial. 3. Desmatamento. 4. Deep Learning I. Gris, João Marcos. II. Souza, Victor Feitosa de Carvalho. III. Goldschmidt, Ronaldo Ribeiro. IV. Draeger, Vitor Henrique Pereira. V. Título. VI. Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

EDUARDO CESAR NOGUEIRA COUTINHO
JOÃO MARCOS GRIS
1º Ten VICTOR FEITOSA DE CARVALHO SOUZA

DEEP LEARNING NA ANÁLISE DE IMAGENS PARA DETECÇÃO DE REGIÕES DE DESMATAMENTO

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Prof. Ronaldo Ribeiro Goldschmidt - D.Sc.

Co-Orientador: Maj Vitor Henrique Pereira Draeger - M.Sc.

Aprovado em 09 de Outubro de 2017 pela seguinte Banca Examinadora:

Prof. Ronaldo Ribeiro Goldschmidt - D.Sc. do IME - Presidente

Maj Vitor Henrique Pereira Draeger - M.Sc. do IME

Prof. Carla Liberal Pagliari - Ph.D. do IME

Rio de Janeiro
2017

AGRADECIMENTOS

Agradecemos às pessoas que nos incentivaram e possibilitaram esta oportunidade de expandir nossos horizontes.

Aos nossos familiares e mestres.

Em especial, aos nossos orientadores Prof. Ronaldo Goldschmidt e Maj. Vitor Draeger por todo o apoio no desenvolvimento do projeto e pelos conhecimentos compartilhados.

“A Ciência é uma irmã caçula (talvez bastarda) da Arte. ”

CÉSAR LATTES

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

IA	Inteligência Artificial
ML	*Machine Learning*
RNA	Rede Neural Artificial
CNN	*Convolutional Neural Network*
MLP	*Multi Layer Perceptron*
SGD	*Stochastic Gradient Descent*
UC	*User Case*

RESUMO

O desmatamento é um problema de difícil tratamento uma vez que a recuperação natural de florestas pode levar centenas de anos, quando possível. Desse modo, a identificação de focos de desmatamento pode auxiliar no combate e prevenção do avanço das áreas desmatadas. Nesse trabalho, foi desenvolvida uma ferramenta que viabiliza a criação de modelos de redes neurais para identificação automatizada de regiões desmatadas. Foram treinados modelos de redes convolucionais, que conseguiram obter acurácia acima de 80% na predição de áreas desmatadas. A plataforma desenvolvida permite a customização de parâmetros de treinamento e disponibiliza uma interface para visualização interativa dos focos de desmatamento estimados pelo modelo.

ABSTRACT

Deforestation is a problem of difficult treatment since the natural recovery of forests can take hundreds of years, when possible. In this way, the identification of deforestation outbreaks can help in prevention and battle against growth of deforested areas. In this work, a tool was developed that enables the creation of neural network models for automated identification of deforested regions. Convolutional network models were trained, which were able to obtain an accuracy of more than 90 % in the prediction of deforested areas. The developed platform allows the customization of training parameters and provides an interface for the interactive visualization of deforestation foci estimated by the model.

1 INTRODUÇÃO

De 1990 para 2015, houve uma perda de $1,3$ milhão de km^2 de área florestada, o que corresponde a 3% da área total existente no mundo(WORLD-BANK).

No Brasil, o problema é bastante importante, pois possuímos grande parte do território amazônico. O desmatamento na Amazônia Legal tem se mantido acima de 20.000 km^2 por ano (SOARES, 2012). Além de se pesquisar maneiras de reduzir estes números por meio da educação, prevenção e conscientização, existe também uma preocupação em identificar as áreas mais afetadas.

A aprendizagem profunda tem sido empregada para resolver problemas de grande complexidade e vem obtendo bons resultados (SILVER et al., 2016) (SIMONYAN; ZISSERMAN, 2014). Além disso, a Visão Computacional é uma das áreas na qual a aprendizagem profunda tem conseguido gerar grande impacto (DENG; YU, 2014). Por outro lado, a área de Sensoreamento Remoto também vem contribuindo para a obtenção de métodos mais efetivos de controle ambiental (PHYS-ORG). No entanto, aplicações que se apoiem em todos esses campos de estudo para detecção de desmatamento ainda são limitadas.

1.1 MOTIVAÇÃO

Em razão do problema do desmatamento, há uma demanda por ferramentas que possam automatizar este processo. Com todas as facilidades geradas a partir de imagens obtidas por satélite, um sistema que utilize técnicas de inteligência computacional para indicar áreas com maior grau de desmatamento seria de grande utilidade para entidades públicas ou até mesmo privadas que se preocupem com controle florestal.

1.2 OBJETIVO

O objetivo dessa pesquisa é desenvolver uma plataforma web que possibilite o treino customizável de algoritmos de aprendizado de máquina baseados em aprendizagem profunda, permitindo a identificação de regiões de desmatamento a partir de dados de imagens de satélites georreferenciadas. Além disso, deseja-se que a ferramenta promova a visualização das regiões de desmatamento segundo a classificação sugerida pelos algoritmos.

1.3 JUSTIFICATIVA

Plataformas para identificação de desmatamento são limitadas. As que estão disponíveis no mercado e no meio acadêmico não explicam detalhadamente a abordagem utilizada ou não permitem qualquer tipo de customização por parte do usuário.

Um trabalho de grande impacto (HANSEN et al., 2013) mapeou as regiões de desmatamento no mundo com uma resolução espacial de 1 arco de segundo por pixel (aproximadamente $30m^2$ por pixel, próximo ao equador). No entanto, o modelo utilizado não é público, e com isso não se sabe a capacidade de generalização do método nem se pode ajustá-lo para trabalhar com imagens de outras resoluções ou de regiões específicas; muito menos pode-se customizar os parâmetros do modelo e comparar resultados.

1.4 CONTRIBUIÇÕES ESPERADAS

Espera-se desenvolver, a partir desse trabalho, uma forma automatizada de se determinar regiões de desmatamento utilizando imagens de satélite de uma determinada região em tempos distintos. Além disso, deseja-se que a plataforma possua uma interface amigável e a infraestrutura adequada para realizar o aprendizado a partir dos dados de satélite, contando com funcionalidades que permitam customização dos parâmetros envolvidos no processo de aprendizado e permitindo a visualização dos resultados. A plataforma será capaz de persistir os dados de satélite e os modelos treinados, de forma que seja possível comparar os resultados dos estimadores.

1.5 MÉTODO

Para que o objetivo fosse alcançado com maior confiabilidade, realizou-se uma prova de conceito para se determinar a viabilidade da extração de conhecimento sobre desmatamento das bases anotadas de dados que estavam à disposição. Avaliou-se, nessa prova de conceito, o desempenho de um classificador linear e de uma rede neural convolucional aplicados para determinar o grau de florestamento de uma região em tempos distintos, e assim averiguar se houve ou não desmatamento naquela área. Após isso, foi desenvolvido um protótipo que possibilitou o treino de classificadores para inferência do grau de florestamento, além da capacidade de visualização de áreas desmatadas. Finalmente, o sistema permitiu a comparação de diversos modelos de classificador e análises espaço-temporais de regiões de desmatamento.

1.6 ESTRUTURA DA MONOGRAFIA

O capítulo 2 é direcionado à fundamentação teórica, onde os conceitos de aprendizagem automática, redes neurais e aprendizagem profunda são explicados e exemplificados. Os fundamentos dos métodos de otimização geralmente utilizados em aprendizagem profunda são apresentados logo em seguida.

O capítulo 3 descreve os principais trabalhos desenvolvidos na área de aprendizagem profunda aplicada à visão computacional e na área de sensoriamento remoto para detecção de desmatamento.

O capítulo 4 descreve a solução proposta e exibe a descrição conceitual do protótipo desenvolvido, apresentando a modelagem de casos de uso e as telas projetadas para a plataforma. Em seguida, exibe-se a prova de conceito que verifica a factibilidade da classificação das regiões de desmatamento a partir de aprendizagem profunda, além da especificação do protótipo.

O capítulo 5 apresenta os resultados da análise da acurácia dos classificadores de aprendizagem profunda para identificação do desmatamento, através de uma série de experimentos.

Finalmente, o capítulo 6 apresenta uma visão geral do trabalho desenvolvido, comentando possíveis melhorias e funcionalidades futuras.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 APRENDIZADO DE MÁQUINA

2.1.1 VISÃO GERAL

A Inteligência Artificial (IA) é o termo designado ao ramo da Ciência da Computação que estuda a criação de máquinas ou algoritmos que desempenham funções que usualmente são designadas ao seres humanos (ABU-MOSTAFA et al., 2012). O desafio da inteligência artificial se acentua em problemas que podem ser facilmente resolvidos por pessoas, mas são difíceis de serem descritos formalmente, como reconhecimento de imagens (LOWE, 1987) e interpretação de textos (SCHAPIRE; SINGER, 2000), por exemplo.

O Aprendizado de Máquina (ou ML - *Machine Learning*) é uma área da IA que explora o estudo e construção de algoritmos que podem aprender a partir dos seus erros e fazer previsões sobre os dados (SCHÜRMANN, 1996). Os problemas de ML podem ser divididos em três grandes áreas (ABU-MOSTAFA et al., 2012): Aprendizagem Supervisionada, Aprendizagem Não-Supervisionada e Aprendizagem por Reforço.

Na Aprendizagem Supervisionada, o sistema de aprendizagem recebe entradas e saídas referentes a um determinado conjunto de dados e, a partir deles, encontra uma função capaz de mapear as entradas nas saídas. O Aprendizado Supervisionado pode ainda ser subdividido em Classificação ou Regressão (ABU-MOSTAFA et al., 2012). Na Classificação, o conjunto de saídas tem um número finito de elementos, denominados classes. Em um problema de regressão, por outro lado, o conjunto de saídas é contínuo, conforme mostra a figura 2.1.

Na aprendizagem Não-Supervisionada, não há saídas previamente relacionadas com as entradas, cabendo ao sistema encontrar padrões nos dados, separando-os em categorias.

Na Aprendizagem por Reforço, o sistema de aprendizagem desenvolve e aperfeiçoa seu algoritmo na interação com um determinado meio para desempenhar certa tarefa. Por exemplo, um robô bípede que aprende a caminhar otimizando um algoritmo interno à medida que tenta executar essa atividade realiza um aprendizado por reforço. A figura 2.2 exibe hierarquicamente as divisões dos diversos campos da Inteligência Artificial.

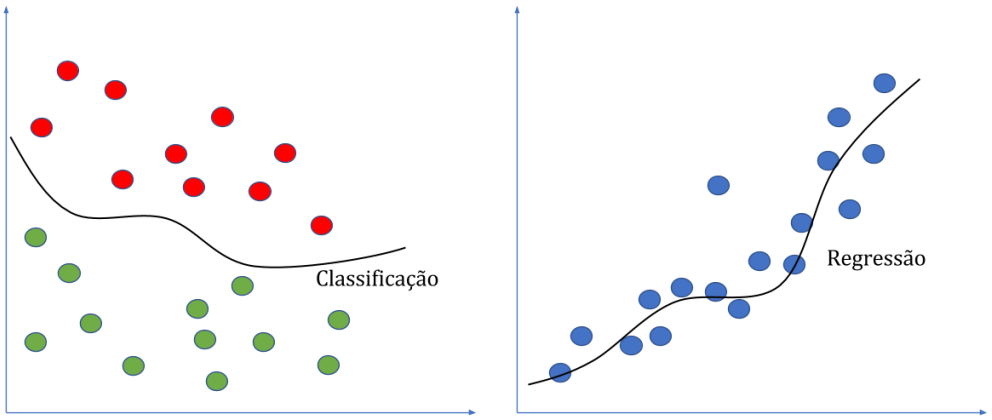


FIG. 2.1: Exemplo de funções de regressão e classificação

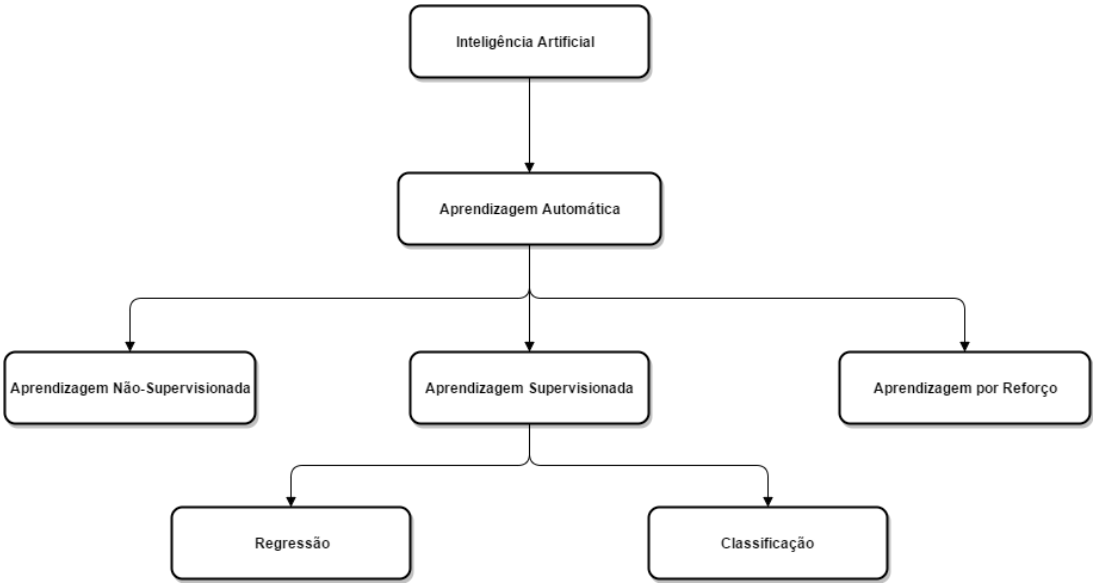


FIG. 2.2: Esquema hierárquico dos campos da inteligência artificial

2.1.2 O PROBLEMA DA APRENDIZAGEM

A figura 2.3 representa de forma esquemática o processo de aprendizagem em *Machine Learning*. O problema central consiste em aproximar uma função, denominada função objetivo.

$$f : X \mapsto Y \tag{2.1}$$

Na equação 2.1, X corresponde ao conjunto de entradas possíveis do problema e Y se refere ao conjunto de possíveis saídas. O conjunto de dados, chamado de conjunto de treino ($\mathcal{D}$), é constituído de uma coleção, onde $\mathcal{D} = (x_n, y_n)$, com $n \in \{1, 2, \ldots, N\}$ para os quais é sabido que $f(x_n) = y_n$. Além disso, as entradas $x_1, \ldots, x_N$ estão associadas a uma distribuição de probabilidade desconhecida $P(x)$, que representa a probabilidade associada à aparição de uma variável x como entrada para o problema.

Para aproximar a função objetivo, considera-se um conjunto de hipótese H (Eq. 2.2), que é constituído de uma coleção de funções da qual se deve escolher uma função g que se aproxime da função objetivo desconhecida.

$$\mathcal{H} = \{h_1, h_2, ..., h_M\} \tag{2.2}$$

Para realizar essa escolha, define-se uma função de erro tanto para o conjunto de treino quanto para o conjunto de entradas possíveis (E_{in} e E_{out}, respectivamente). No caso de Classificação, é comum definir essas funções da seguinte forma:

$$E_{in}(h) = \frac{1}{N} \sum_{n=1}^{N} [\![h(x_n) \neq f(x_n)]\!] \tag{2.3}$$

$$E_{out}(h) = P[h(x) \neq f(x)] \tag{2.4}$$

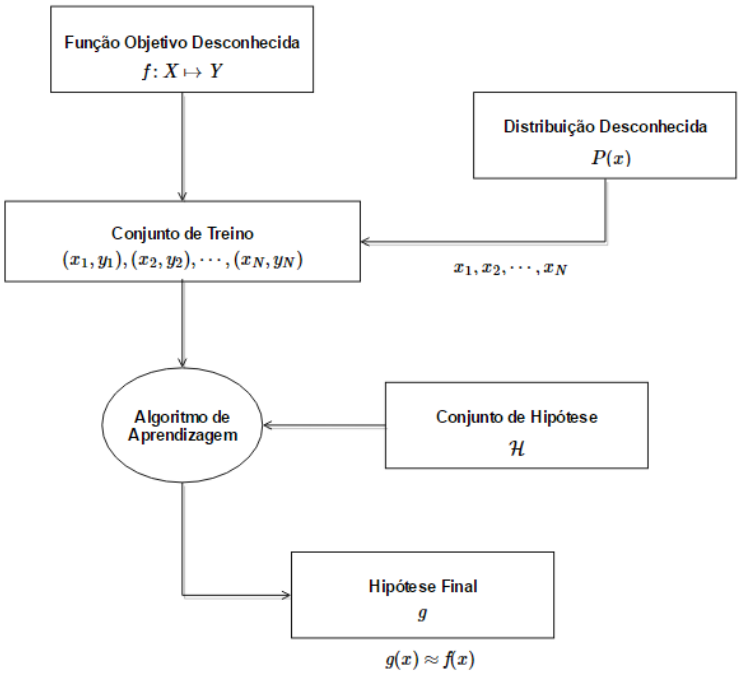


FIG. 2.3: Diagrama do processo de aprendizagem - Adaptado de Abu-Mostafa et al. (2012)

No problema da aprendizagem, busca-se minimizar $E_{in}(h)$ em vez de $E_{out}(h)$, uma vez que não se conhece $f(x)$ para $x \notin \mathcal{D}$. A desigualdade de Hoeffding (Eq. 2.5) garante que minimizar $E_{in}(h)$, quando o conjunto de treino é suficientemente grande, faz com que $E_{out}(h)$ também seja pequeno, com alta probabilidade.

Portanto, o problema do aprendizado é factível. Ou seja, pode-se aproximar a função objetivo com um determinado grau de certeza a partir da minimização do erro em um conjunto de amostra, garantindo que a função g seja capaz de generalizar o aprendizado obtido no conjunto de dados (D).

$$P[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N} \text{ , para todo } \epsilon > 0 \tag{2.5}$$

2.2 REDES NEURAIS ARTIFICIAIS

Uma rede neural artificial (RNA) é um modelo computacional utilizado em Aprendizado de Máquina. Ela é composta de uma estrutura de funções de transferências chamadas neurônios (*neurons*), que são aplicadas sucessivas vezes em um conjunto de entradas, também chamadas de características (*features*), gerando um conjunto de saídas, que podem ser utilizadas para realizar classificação ou regressão (FACELI, 2011).

A estrutura de uma rede neural é organizada em camadas. A primeira camada é denominada camada de entrada, a última camada é chamada de camada de saída e as intermediárias são as camadas ocultas.

Cada *neuron* recebe um conjunto de saídas da camada anterior e, idealmente, realiza operações que identificam padrões a partir dessas entradas, propagando-os para a próxima camada. Matematicamente, um *neuron* é composto de um conjunto de entradas $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ provenientes da camada anterior, um vetor de pesos associados $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ e uma função de transferência g, também chamada de função de ativação. A saída y de um neurônio é determinada pela relação abaixo.

$$a(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^T \mathbf{x} + \mathbf{b} \tag{2.6}$$

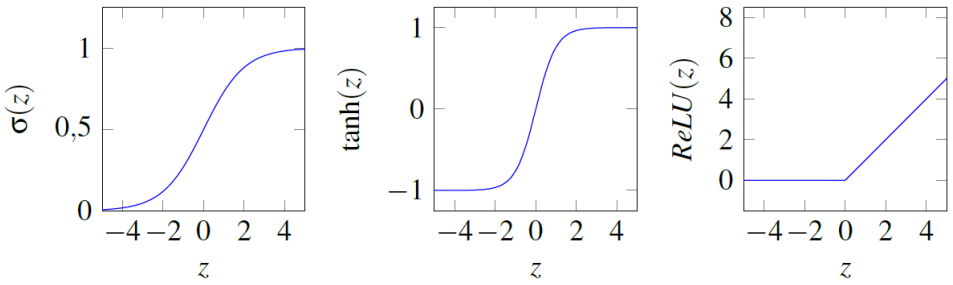
$$y = h(\mathbf{x}) = g(a(\mathbf{x})) \tag{2.7}$$

O termo b é denominado fator de polarização. Há diversas alternativas para a escolha da função de ativação. Dentre elas, destacam-se a função sigmoide logística (Eq. 2.8), sigmoide tangente (Eq. 2.9) e a retificada linear (Eq. 2.10). A figura 2.4 exibe os gráficos dessas funções.

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.8}$$

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.9}$$

$$g(z) = max(0, z) \tag{2.10}$$



(a) Logística. (b) Hyperbólica tangente. (c) Retificada linear.

FIG. 2.4: Funções de ativação

As equações apresentadas podem ser adaptadas para uma rede de L camadas ocultas, com vários neurônios por camada.

Seja $\mathbf{W}_k$ a matriz de pesos entre as camadas $k-1$ e k, $\mathbf{h}_0(\mathbf{x}) = \mathbf{x}$, representando as entradas da rede ($\mathbf{x} = (x_0, x_1, ..., x_D)$). Para $k \in \{1, 2, \ldots, L, L+1\}$, definimos:

$$\mathbf{h}_k(\mathbf{u}) = (h_{k,0}(u_0), h_{k,1}(u_1), ..., h_{k,m_k}(u_{m_{k-1}})) \tag{2.11}$$

$$\mathbf{a}_k(\mathbf{x}) = \mathbf{b}_k + \mathbf{W}_k \mathbf{h}_{k-1}(\mathbf{x}) \tag{2.12}$$

$$\mathbf{h}_k(\mathbf{x}) = \begin{cases} \mathbf{g}(\mathbf{a}_k(\mathbf{x})), & \text{se } k \neq L+1 \\ \mathbf{o}(\mathbf{a}_k(\mathbf{x})), & \text{se } k = L+1 \end{cases} \tag{2.13}$$

Dessa forma, para um vetor de entradas $\mathbf{x}$, a rede neural artificial é definida simplesmente como uma função $f(\mathbf{x}) : R^D \mapsto R^C$:

$$f(\mathbf{x}) = \mathbf{h}_{L+1}(\mathbf{x}) \tag{2.14}$$

Especificamente para o problema de classificação, costuma-se utilizar a função *softmax* (IAN GOODFELLOW, 2016):

$$o(a_{L+1,i}) = \frac{e^{a_{L+1,i}}}{\sum_{j=1}^{C} e^{a_{L+1,j}}} \tag{2.15}$$

Sendo $o(a_{L+1}, i)$ interpretado como a probabilidade de x ser classificado na categoria de índice i.

A figura 2.5 mostra o esquema da rede neural definida com a notação adotada.

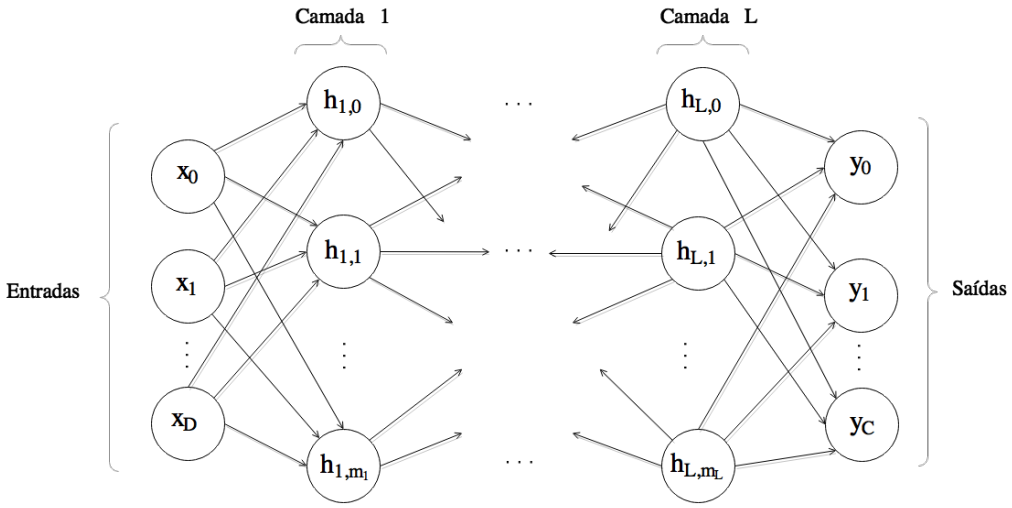


FIG. 2.5: Esquema das camadas de uma rede neural

O objetivo do processo de treinamento é minimizar uma função $\mathbf{J}(\boldsymbol{\theta})$ de erro sobre um conjunto de dados $\{(\mathbf{x}_t, \mathbf{y}_t)\}$, $t \in \{1, 2, \ldots, T\}$, onde $\boldsymbol{\theta}$ representa os vetores de parâmetros da RNA.

$$\mathbf{J}(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T} J(f(\boldsymbol{x}_t, \boldsymbol{\theta}), \boldsymbol{y}_t) + \frac{\lambda}{2T}\sum_{k}\sum_{l} W_{k,l}^2 \tag{2.16}$$

J é uma função de custo, usualmente erro quadrado médio ou entropia cruzada. O termo $\frac{\lambda}{2T}\sum_k \sum_l W_{k,l}^2$ é o custo de regularização, sendo λ o parâmetro de regularização. Quanto maior o valor de λ, maior será o erro atribuído aos pesos da RNA. Sendo assim, o parâmetro define o nível de restrição do conjunto de hipótese, ou grau de adequação da aprendizagem às peculiaridades dos dados de treino.

Diversos algoritmos podem ser utilizados para o problema de otimização da minimização da função de erro. No entanto, por questões práticas, o gradiente descendente (BURGES et al., 2005) e suas diversas variações são amplamente utilizadas.

2.3 APRENDIZAGEM PROFUNDA

Aprendizagem Profunda (*Deep Learning*) é o termo utilizado para se referir a redes neurais com grande número de camadas, possuindo uma estrutura interna capaz de operar sobre as entradas e compor hierarquicamente *features* intermediárias para a representação do dado. Essa técnica tem sido amplamente utilizada, especialmente em problemas onde é difícil extrair analiticamente *features* dos dados (IAN GOODFELLOW, 2016).

Um exemplo de rede neural com esse tipo de arquitetura é a rede neural convolucional (KRIZHEVSKY et al., 2012), que é um tipo de rede neural inspirada no funcionamento do córtex visual. Esse tipo de rede costuma ser utilizada em aprendizado sobre conjuntos de imagens e vídeos. Seus neurônios são conectados localmente; ou seja, ao invés de cada neurônio se conectar a todos elementos da camada anterior, ele pode se conectar a um conjunto restrito, que no caso das imagens são os conjuntos de pixels próximos. A imagem 2.6 ilustra a arquitetura de uma rede neural convolucional utilizada para a classificação de uma imagem.

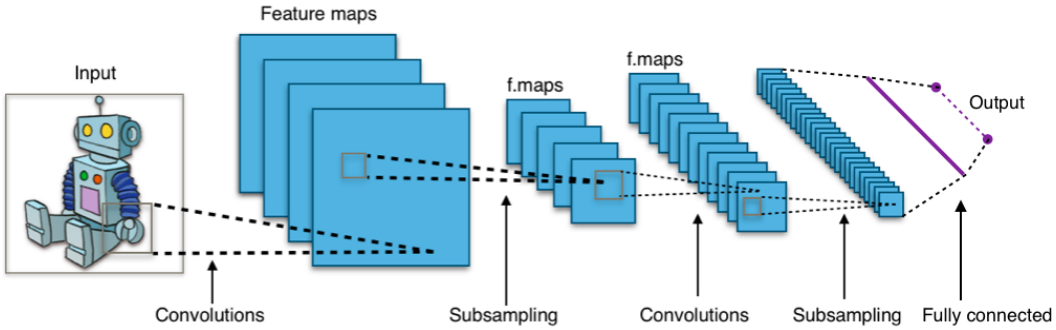


FIG. 2.6: Exemplo de uma rede neural convolucional - Figura extraída de Wikiwand (2017)

Em uma rede neural convolucional há quatro tipos de camadas que são geralmente empregadas (KRIZHEVSKY et al., 2012): camadas de convolução (*convolutional layer*), camadas de subamostragem (*subsampling layer ou max-pooling*), camadas de desligamento (*dropout*) e camadas totalmente conectadas (*fully connected layer*).

Uma camada de convolução é um conjunto de *neurons* dispostos em *grids* retangulares, onde cada um deles realiza uma convolução de um conjunto de saídas da camada anterior, utilizando uma matriz de pesos a ser otimizada. Uma camada de subamostragem tem por objetivo agregar as saídas de uma camada de convolução, tornando-a mais robusta a transformações das imagens, como translação. As camadas de desligamento são camadas de regularização, que recebem como entrada as saídas dos neurônios anteriores e

com uma determinada probabilidade tem como saída zero. As camada totalmente conectadas são camadas onde todos os *neurons* recebem todos os elementos da camada anterior como entrada, gerando descritores de *features* da imagem, que auxiliam na classificação na última camada, geralmente compostas da função *softmax*.

A convolução discreta entre duas funções a, b, definidas nos inteiros, pode ser definida da seguinte forma:

$$a(n) * b(n) = \sum_{i=-\infty}^{\infty} a(i)b(n-i) \tag{2.17}$$

Uma imagem pode ser interpretada como uma função (a) definida nos inteiros, onde $a(n)$ é igual ao valor do pixel de ordem n, quando $0 < n \leq N$ e $a(n) = 0$, caso contrário.

A convolução discreta pode ser interpretada como um produto escalar que é uma métrica de similaridade entre vetores. Dessa forma, a convolução $a(n) * b(n)$ assumirá valores de acordo com o grau de semelhança entre a e b em torno de n, conforme mostra a figura 2.7. A função $b(n)$ é descoberta no processo de aprendizagem e pode ser interpretada como um padrão a ser verificado na imagem.

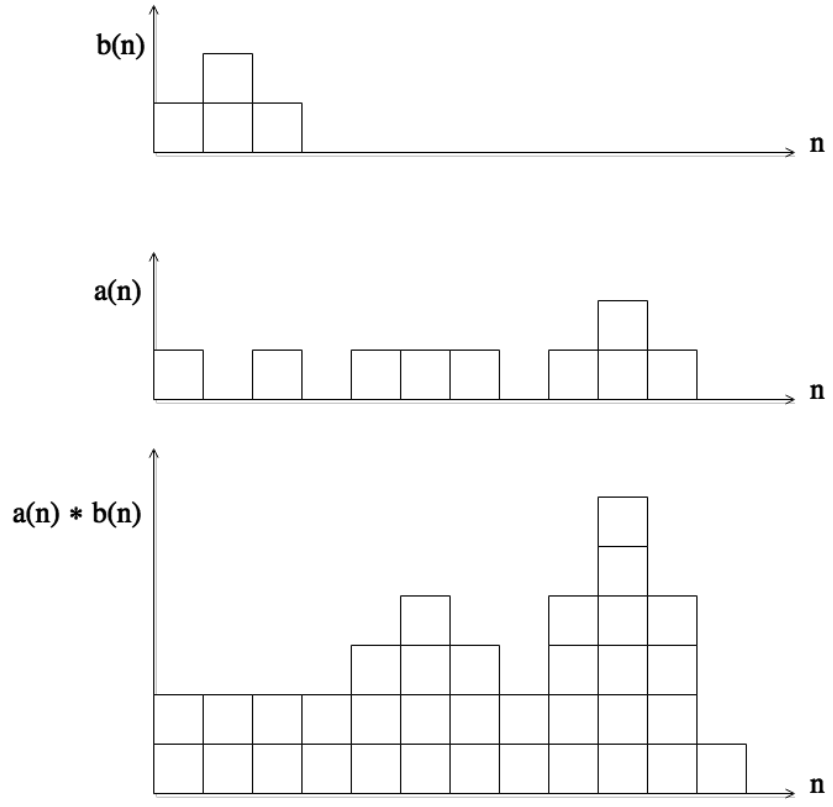


FIG. 2.7: Exemplo de convolução discreta e identificação de padrões

O *max polling* é o processo pelo qual uma função (ou imagem) é divida em regiões de

mesmo tamanho, extraindo-se o maior valor dessas regiões. Isso torna a saída menor do que a entrada (o que é desejável, pois diminui-se o número de parâmetros ao se otimizar nas próximas camadas da rede) e torna a aprendizagem mais resistente a certas transformações nos dados, como a translação, uma vez que a saída não será alterada caso haja pequenas perturbações nos pixels da imagem. A figura 2.8 exemplifica o processo realizado em uma matriz de duas dimensões.

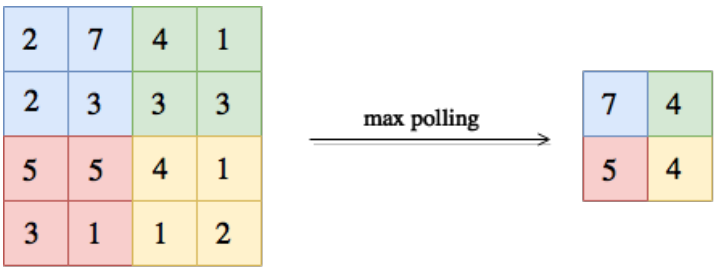


FIG. 2.8: Exemplo de *max polling* em uma imagem em escala de cinza

Uma camada de *dropout* torna, com uma determinada probabilidade, a saída de um *neuron* dessa camada zero. Ter uma saída nula em uma camada oculta da rede diminui o tempo de computação no processo de treino. Além disso, as camadas de *dropout* regularizam a rede, uma vez que eliminar estocasticamente saídas das camadas ocultas adiciona um ruído no processo de treino, o que faz com que a otimização gere uma rede com maior capacidade de generalização.

2.4 OTIMIZAÇÃO

2.4.1 CONCEITOS FUNDAMENTAIS

Treinar uma rede neural é uma tarefa de otimização extremamente custosa que pode durar horas, dias ou até meses (IAN GOODFELLOW, 2016). Dada esta dificuldade, foram criadas técnicas de otimização especialmente para resolver este problema.

O problema de otimização em redes neurais é um problema que pode ser enunciado como "Achar os parâmetros θ que minimizam uma função custo $J(\theta)$".

Algoritmos de otimização que usam todo o conjunto de treino são chamados de algoritmos em batch ou em lote ou ainda métodos determinísticos, pois o processo de treinamento utiliza todos os exemplos simultaneamente em um grande lote. Algoritmos que utilizam um exemplo de treinamento por vez são nomeados de algoritmos online ou estocásticos (IAN GOODFELLOW, 2016).

2.4.2 ALGORITMOS DE OTIMIZAÇÃO

O algoritmo mais utilizado e conhecido na área de Deep Learning é o *Stochastic Gradient Descent* (SGD) ou Descida do Gradiente Estocástica (IAN GOODFELLOW, 2016).

Algorithm 1 Stochastic Gradient Descent - Adaptado de Ian Goodfellow (2016)

1: **entrada**: taxa de aprendizado ϵ_k
2: **entrada**: parâmetros iniciais θ
3: Enquanto $J(\theta)$ não estiver suficientemente próximo de um mínimo:
4: amostre um minibatch de m exemplos do conjunto de treino $\{x^{(1)}, ..., x^{(m)}\}$
5: Calcule a estimativa do gradiente: g
6: $\theta \leftarrow \theta - \epsilon g$
7: Fim

Por outro lado, uma série de esforços foram feitos para elaborar algoritmos mais robustos que o SGD (IAN GOODFELLOW, 2016). Surgem então os algoritmos com taxa de aprendizado variável. Dentre eles, um dos mais utilizados é o Adagrad. SGD realiza a mesma atualização dos parâmetros θ, já que todos os θ_i tem a mesma taxa de aprendizado η. Já Adagrad possui uma taxa de aprendizado diferente para cada parãmetro em cada instante de tempo t. Faremos agora uma descrição das equações que regem o Adagrad.

Seja $g_{t,i}$ o gradiente da função custo $J(\theta)$ em relação ao parâmetro θ_i e ao instante de tempo t.

$$g_{t,i} = \nabla_\theta J(\theta_i) \tag{2.18}$$

A atualização dos parâmetros no SGD seria dada da seguinte forma:

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i} \tag{2.19}$$

No Adagrad, a atualização dos parâmetros modifica a taxa geral η de aprendizado para cada parâmetro θ_i em cada instante t.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i} \tag{2.20}$$

G_t é uma matriz diagonal cujo elemento i, i é a soma dos quadrados dos gradientes com respeito a θ_i até o instante de tempo t.O valor ϵ é um fator colocado no denominador para suavizá-lo (não zerar o denominador). Usualmente é da ordem de 10^{-8} (DUCHI et al., 2011).

A principal vantagem do *Adagrad* é não necessitar de ajuste manual na taxa de aprendizado; usualmente é inicializada com $0,01$ (IAN GOODFELLOW, 2016). Sua

principal desvantagem é o acúmulo de gradientes no denominador, pois, com o tempo, a taxa de aprendizado acaba tornando-se muito próxima de zero, impedindo o algoritmo de continuar o aprendizado.

Uma variação do *Adagrad*, denominada *Adadelta* (ZEILER, 2012), é um método de otimização que busca reduzir a agressividade do *Adagrad*. Nesse método, a taxa de aprendizagem é reduzida monotonicamente à medida que as iterações são realizadas. O método *Adadelta* restringe a janela de gradientes acumulados a um valor fixo w, que deve ser escolhido pelo usuário do método.

RMSprop é um método de otimização de taxa adaptativa não publicado(IAN GOODFELLOW, 2016), proposto por Geoff Hinton em seu curso online(MOOC) disponibilizado pela plataforma Coursera(HINTON). *RMSprop* foi desenvolvido independentemente do *Adadelta* praticamente no mesmo período(IAN GOODFELLOW, 2016). Ambos buscam reduzir a agressividade intrínseca ao *Adagrad*. O *RMSprop* algoritmo usa uma média móvel exponencial com valor sugerido de $\gamma = 0.9$.

3 TRABALHOS RELACIONADOS

3.1 APRENDIZAGEM PROFUNDA

3.1.1 *IMAGENET CLASSIFICATION WITH DEEP CONVOLUTIONAL NEURAL NETWORKS*

No trabalho Krizhevsky et al. (2012), um dos muitos publicados na área de Visão Computacional com aplicação em Aprendizagem Profunda, os autores exploraram dois repositórios públicos de imagens ImageNet LSVRC-2010 e LSVRC-2012 (IMAGENET, 2017), que podem ser considerados como referências de desempenho para modelos de reconhecimento de imagens, uma vez que foram utilizados em competições de alto nível.

Os repositórios foram constituídos de 1.2 milhões de imagens designadas para treino, 50.000 para validação e 150.000 para teste, todas com alta resolução.

O modelo preditivo empregado foi uma rede neural convolucional com 8 camadas, sendo 5 de convolução e 3 totalmente conectadas, alcançando as menores taxas de erros registradas até então para esses conjuntos de dados. Para o LSVRC-2010, obteve-se um erro de $37,5\%$ e $17,0\%$ para o erro top-1[1] e erro top-5[2], respectivamente. Já para o LSVRC-2012, obteve-se um erro de $36,7\%$ para o erro top-1 e 15.3% para o erro top-5.

3.1.2 *GOING DEEPER WITH CONVOLUTIONS*

Um outro trabalho explorou a aplicação de aprendizagem profunda em Visão Computacional, com grande impacto (SZEGEDY et al., 2014), construindo um modelo de aprendizagem que venceu a competição LSVRC-2012 de rede neural convolucional com 22 camadas, com o auxílio da arquitetura denominada Inception (SZEGEDY et al., 2016). O classificador desenvolvido obteve erro top-5 de $6,67\%$, enquanto o segundo lugar da competição obteve 7.32%.

Esse trabalho mostrou o quão precisa a aplicação de redes neurais convolucionais, quando associada a uma engenharia cuidadosa na construção da arquitetura da rede e poder computacional disponível, pode ser em reconhecimento de imagens.

[1]Nessa métrica de erro, considera-se uma classificação errada aquela que não corresponde à saída de maior probabilidade sugerida pela rede

[2]Nessa métrica de erro, considera-se uma classificação errada aquela que não corresponde às 5 saídas de maior probabilidade

A figura 3.1 exibe a rede neural vencedora da competição.

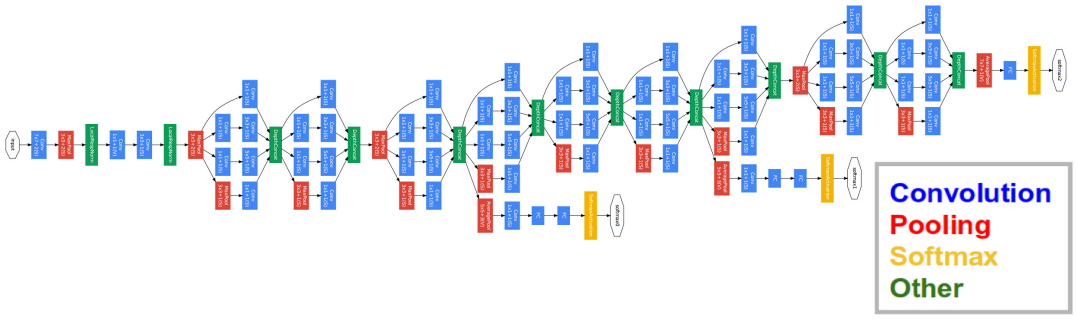


FIG. 3.1: Rede neural convolucional googlenet[3]

3.2 DETECÇÃO DE DESMATAMENTO

3.2.1 *HIGH-RESOLUTION GLOBAL MAPS OF 21ST-CENTURY FOREST COVER CHANGE*

O trabalho, publicado na revista *Science* em novembro de 2013 (HANSEN et al., 2013), abarca um amplo estudo geográfico de áreas florestais. Foram utilizados dados da Terra obtidos pelo programa de satélites LANDSAT de 2000 a 2012, com uma resolução espacial de 30 metros. Em relação aos trabalhos prévios relacionados ao tema, o artigo inovou em ser espacialmente explícito; quantificar ganho e perda de florestamento; fornecer informação anual de perda e quantificação de tendências relativas à perda de florestamento; ser estruturado a partir de uma abordagem internamente consistente que se adapta às nuances de diferentes definições, métodos e entradas de dados.

Quanto ao método do trabalho, foram utilizados parâmetros de cobertura florestal (representação da intensidade de área florestal numa determinada região), de perda e de ganho de florestamento. A partir de tais variáveis, foi possível analisar a evolução das regiões florestais por todo o mundo e correlacionar práticas exploratórias ou de proteção ambiental com seus efeitos no florestamento observado.

Como resultados do artigo, ressalta-se que o único domínio climático a exibir uma tendência foi a zona tropical, com perda de florestamento aumentando 2101 quilômetros quadrados por ano. Além disso, confirmou-se a documentada redução no desmatamento no Brasil, enquanto que nas regiões da Indonésia, Malásia, Paraguai (entre outras) foram observados aumentos nos índices de perda florestal.

[3]Disponível no endereço http://deeplearning.net/tag/googlenet/

3.2.2 *ANALYSIS OF SATELLITE IMAGES TO TRACK DEFORESTATION*

O trabalho apresenta uma solução para o problema de identificação das áreas de florestamento com o auxílio de inteligência artificial (ŠIMIĆ DE TORRES, 2016). Como proposta de estudo, buscou-se aliar o volume de dados disponível nas plataformas de dados georreferenciados Terra-I e Landsat8 com a possibilidade de identificação de áreas de desmatamento através de algoritmos de redes neurais.

Enquanto que o sistema Terra-I[4] apresenta a vantagem de disponibilizar o monitoramento em tempo quase real das áreas de desmatamento, a resolução das imagens disponíveis na plataforma não é grande o suficiente para permitir análise espacial, mas somente temporal das imagens. Com o lançamento do satélite Landsat8, que permitiu um avanço na resolução espacial das imagens para 30m, foi possível uma nova gama de análises espaciais em imagens de satélite, fato em que se baseou o desenvolvimento do trabalho citado.

A metodologia do trabalho baseou-se na utilização de algoritmos de redes neurais convolucionais e Regressão Logística para mapear os pixels existentes nas imagens em três categorias: floresta, solo e nuvem. Através de tais parâmetros, a mudança de classificação de um pixel de *floresta* para *solo* é um indicativo de desmatamento. Pode-se dizer que a existência das regiões de nuvem foi o principal fator responsável pela dificuldade de se criar um modelo satisfatoriamente preciso na tarefa de classificação. Apesar desse empecilho, obtiveram-se resultados consistentes com as observações extraídas do *Global Forest Watch*, um sistema de monitoramento ambiental de grande acurácia.

A principal diferença do presente trabalho em relação ao trabalho citado é que o primeiro se propõe a utilizar aprendizagem profunda na classificação de imagens, enquanto que o segundo limita-se a otimizar as redes neurais a partir de tentativas sucessivas. Nesse sentido, o presente trabalho abarcou um conjunto de algoritmos de otimização em redes neurais como forma de possibilitar ao usuário uma certa flexibilidade na análise dos dados. Por último, o presente trabalho também propôs fornecer um mecanismo interativo de visualização de imagens, recurso não contemplado pelo trabalho referenciado.

[4]Disponível no endereço http://www.terra-i.org

3.2.3 PROJETO DE MONITORAMENTO DO DESMATAMENTO NA AMAZÔNIA LEGAL POR SATÉLITE (PRODES)

De acordo com INPE-OBT, o projeto PRODES é um projeto desenvolvido pelo INPE (Instituto Nacional de Pesquisas Espaciais) em parceria com o Ministério do Meio Ambiente (MMA) e o Instituto Brasileiro do Meio Ambiente e dos Recursos Naturais Renováveis (IBAMA). Realiza o levantamento de dados e monitoramento do desmatamento na Amazônia Legal por corte raso[5] desde 1988, fornecendo dados úteis ao governo brasileiro para o estabelecimento de políticas públicas.

O sistema utiliza, primordialmente, imagens de satélite da classe Landsat, que possuem resolução espacial de 20 a 30 metros e são recolhidas em um intervalo de 16 dias. Esse conjunto de imagens possibilita um acompanhamento em tempo real das áreas de desmatamento, através de mapeamento abrangendo uma área sempre superior a 6,25 hectares. Por meio desses dados, é possível, através de técnicas de análise dos polígonos de desmatamento, obter um relatório anual de perda florestal observada na Amazônia Legal, com especificações para cada estado brasileiro presente na região Amazônica.

O trabalho desenvolvido pelo PRODES possui em comum com o presente trabalho a análise de dados florestais e a classificação de regiões como desmatadas ou não. Em contrapartida, o PRODES não utiliza mecanismos de inteligência artificial, limitando-se a analisar as imagens por meio de modelos físicos. Além disso, o presente trabalho não possui a finalidade de contemplar o monitoramento completo de uma região florestal, mas fornecer meios de detecção de desmatamento em um conjunto de imagens fornecidos pelos usuários do sistema.

[5] *Corte raso* é definido como o processo de remoção total e abrupta de uma região florestal, com a finalidade de reutilização da área para outros fins (agropecuária, hidrelétricas, entre outros.)

4 SOLUÇÃO PROPOSTA

O desmatamento pode ser definido como a transformação de uma região florestada para uma determinada região não florestada, como campos de agricultura e pecuária (CHAKRAVARTY et al., 2012). Sendo assim, a verificação do desmatamento depende da análise de uma região em dois instantes. De forma mais objetiva, o desmatamento é caracterizado como a redução do nível de florestamento, onde o nível de florestamento, por sua vez, pode ser definido como o valor percentual de uma região que é coberta por árvores.

A abordagem adotada para verificar se houve desmatamento em uma certa região é, primeiro, treinar um classificador que consiga predizer o nível de florestamento dessa região. Para que esse processo de aprendizado seja realizado, é necessário dispor de dados anotados para o treino, ou seja, para uma imagem de tamanho considerável (ou um conjunto de imagens), deve-se ter previamente especificado para cada região o valor do nível de florestamento. Com isso, um classificador pode ser treinado sobre a base dados e generalizar o resultado, predizendo para cada ponto de uma imagem o nível de florestamento associado. De forma mais específica, o treino dos classificadores inseridos no sistema seguiu o seguinte procedimento:

a) Segmentação da imagem de treino e da matriz de anotação em submatrizes de dimensão 20 x 20 pixels, com a finalidade de simplificação dos cálculos;

b) Separação das submatrizes em bases de treino (50%) e teste (50%);

c) Para cada submatriz de anotação, calcular a média do nível de florestamento dos pixels contidos na submatriz e discretizar para pertencer ao conjunto $\{0, 1, ..., 9\}$. Os 9 primeiros níveis se referem aos níveis de florestamento contidos nos intervalos $[10i, 10i + 10)$, com $i \in \{0, 1, ..., 8\}$, enquanto que o nível 10 se refere ao intervalo $[90, 100]$;

d) Treino propriamente dito do classificador, de forma a minimizar o erro da função de predição g. A função de erro adotada é dada por:

$$\sum_{i=1}^{N} \frac{[[g(M_i) \neq n_i]]}{N}$$

onde M_i representa a i-ésima submatriz 20x20, n_i representa o nível de florestamento

discretizado médio (conforme definido acima) calculado com base no dado anotado para a i-ésima submatriz e N representa o total de submatrizes contidas na imagem.

A partir da classificação do nível de florestamento, pode-se determinar se houve desmatamento simplesmente verificando se o nível aumentou ou não em uma determinada região. A figura 4.1 exemplifica como uma imagem pode ser subdividida para se verificar as regiões onde houve desmatamento. No exemplo, a mesma célula (em destaque) obteve níveis de florestamento de 6 e 4 em tempos distintos (onde a célula da direita representa um momento posterior em relação à célula da esquerda). Portanto, a região pode ser classificada como região de desmatamento. O mesmo processo é repetido para todas as células da imagem e assim podem-se mapear as regiões desmatadas.

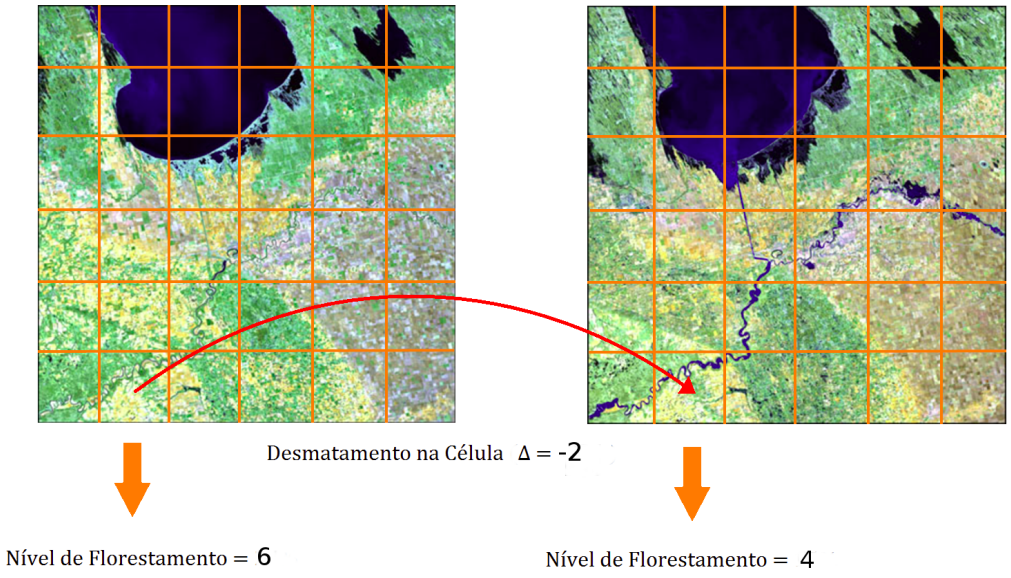


FIG. 4.1: Método de obtenção das regiões de desmatamento

4.1 DESCRIÇÃO CONCEITUAL

Para definir quais funcionalidades deveriam ser implantadas na plataforma de determinação de desmatamento, foram levantados os requisitos funcionais, conforme mostra a tabela abaixo.

A partir dos requisitos funcionais, levantaram-se os casos de uso (junto com suas respectivas descrições resumidas) do protótipo e seu respectivo diagrama, como mostra a figura 4.2.

TAB. 4.1: Requisitos Funcionais

Número	Requisito
1	A plataforma deve permitir o armazenamento de dados de imagens (anotadas ou não)
2	A plataforma deve permitir o treinamento de classificadores sobre um conjunto de imagens anotadas quanto ao nível de florestamento
3	A plataforma deve persistir os classificadores treinados previamente, permitindo sua visualização, edição e exclusão
4	A plataforma deve permitir a classificação das imagens de um conjunto de dados não anotados, através de classificadores previamente cadastrados
5	A plataforma deve exibir as áreas classificadas como desmatadas segundo os classificadores cadastrados

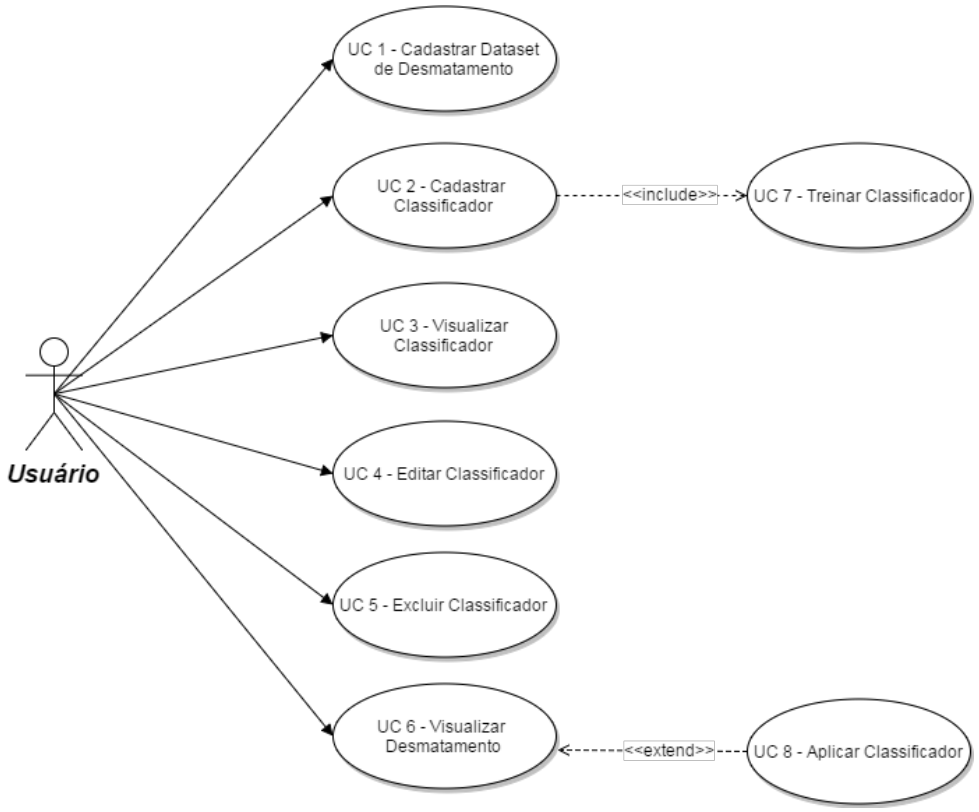


FIG. 4.2: Diagrama de casos de uso

UC 1 (Cadastrar Dataset de Desmatamento) O usuário, após entrar na plataforma, seleciona a opção de "Novo Dataset", sendo redirecionado para uma tela de cadastro. Ele pode, então, submeter um conjunto de dados de imagens georreferenciadas em um formato rígido, que pode estar anotado ou não. Caso o conjunto de dados esteja anotado, este poderá ser usado para treino de classificadores. Caso contrário, poderá ser usado apenas para classificação.

UC 2 (Cadastrar Classificador) O usuário, após entrar na plataforma, seleciona a opção de "Novo Classificador", sendo redirecionado para uma tela de cadastro. Ele pode então cadastrar um classificador selecionando o conjunto de dados previamente cadastrado com o rótulo que deseja utilizar, o tipo de algoritmo e os parâmetros de otimização.

UC 3 (Visualizar Classificador) O usuário, após entrar na plataforma, pode ver uma lista dos classificadores cadastrados, bem como suas respectivas taxas de sucesso.

UC 4 (Editar Classificador) O usuário, após entrar na plataforma, seleciona um item de classificador que deseja editar. Ele então pode renomear o classificador e o dado é persistido na plataforma.

UC 5 (Excluir Classificador) O usuário, após entrar na plataforma, seleciona um item de classificador que deseja excluir. Caso a exclusão seja confirmada, o classificador é retirado da plataforma.

UC 6 (Visualizar Desmatamento) O usuário, após entrar na plataforma, seleciona a opção de "Visualização", sendo então redirecionado para uma tela com o mapa das regiões desmatadas segundo os conjuntos de dados rotulados. Ele pode então selecionar classificadores e datasets (rotulados ou não) para visualizar as regiões de desmatamento sugeridas pelos classificadores e selecionar a opção "Aplicar". O classificador, então, é aplicado nos dois datasets, que representam tempos distintos. Assim a visualização irá fornecer as regiões de desmatamento ao se subtrair os graus de florestamentos encontrados para os dois tempos.

UC 7 (Treinar Classificador) Após o usuário cadastrar um classificador, o treino deste é iniciado. Quando o treino termina, o classificador é atualizado com a função de classificação e pode então ser utilizado para visualização.

UC 8 (Aplicar Classificador) Após o usuário selecionar a opção "Aplicar" na visualização, os classificadores selecionados são aplicados nos conjuntos de dados selecionados, exibindo as regiões desmatadas sugeridas.

Um modelo das telas da plataforma foi projetado para visualização do fluxo de interação do usuário e validação do protótipo. A figura 4.3 exibe esse modelo.

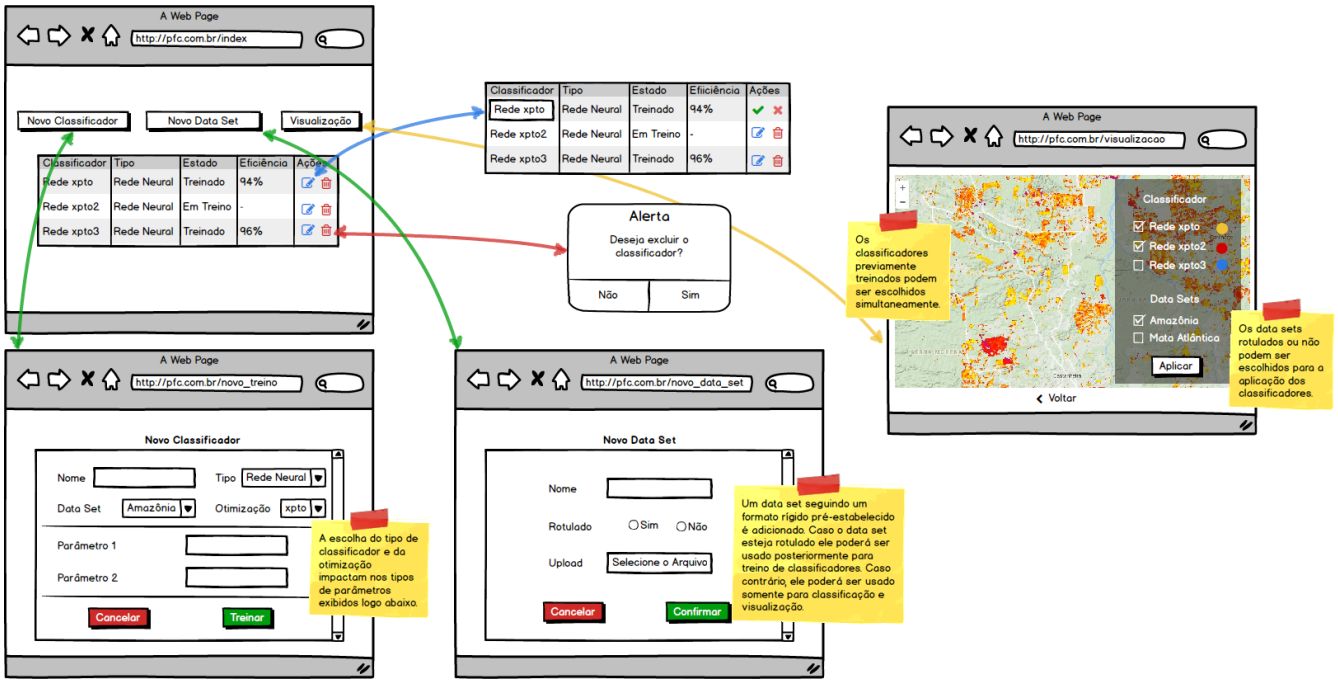


FIG. 4.3: Esquema das telas da plataforma

Para a interação do usuário com a plataforma de treino do classificador (UC 7), foi construído um diagrama de atividades para esquematizar a funcionalidade, conforme mostra a figura 4.4.

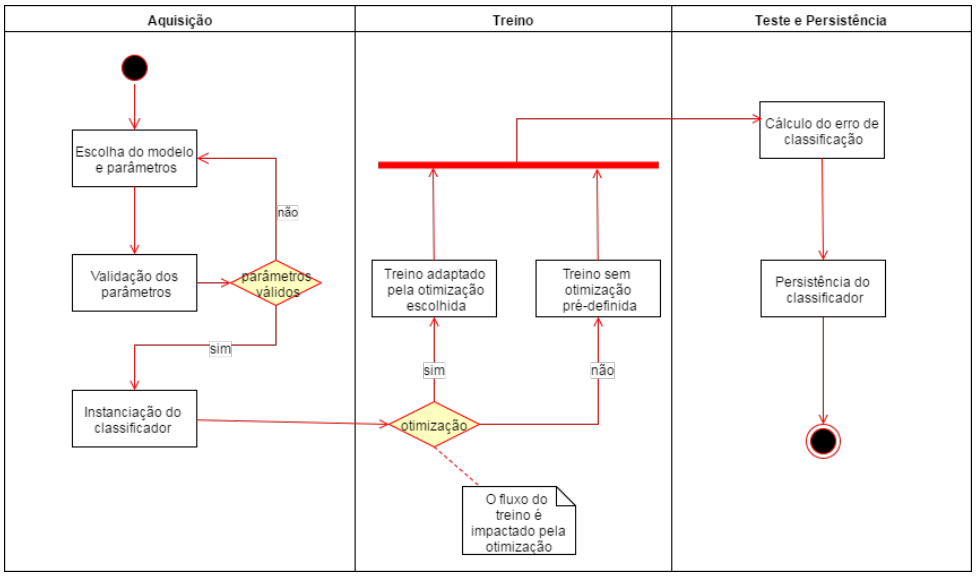


FIG. 4.4: Diagrama de atividades do treino

4.2 PROVA DE CONCEITO

A prova de conceito é uma forma de verificar a viabilidade do projeto. Para tanto, reduziu-se o escopo do problema às partes mais críticas, implementando algumas soluções e analisando-se os resultados.

O escopo escolhido para a prova de conceito é constituído da análise exploratória de dados de florestamento, implementação e aplicação de classificadores para o nível de florestamento.

Os dados utilizados para explorar, treinar e classificar podem ser encontrados em Hansen (2017). Esses dados foram capturados no programa LANDSAT, que é um projeto norte americano de aquisição de imagens de satélite da Terra. Para todo o mundo, em uma resolução espacial de 1 segundo de arco por pixel (aproximadamente 30 m^2 por pixel), há disponível imagens de 2000 e 2014, além do nível de florestamento (tree cover) para o ano de 2000. A figura 4.5 exibe uma região presente nas imagens disponibilizadas e seu respetivo mapeamento quanto ao nível de florestamento.

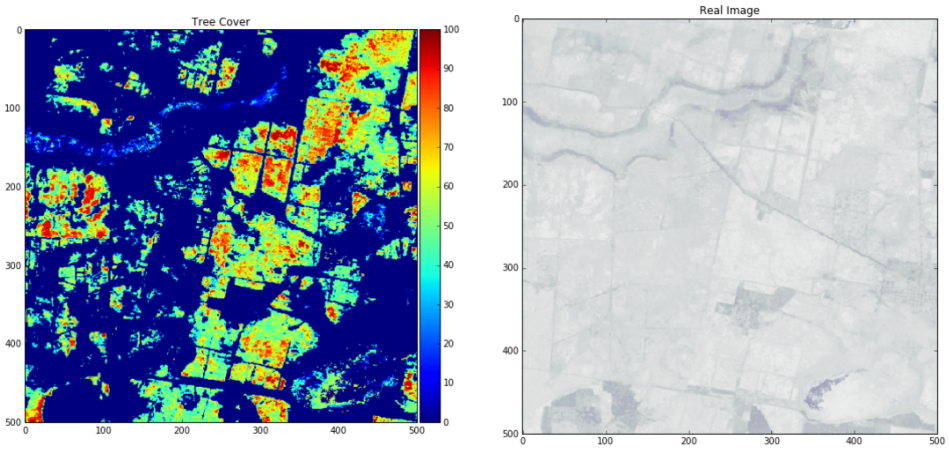


FIG. 4.5: Nível de florestamento e a respectiva imagem de satélite

A partir de uma fração dos dados disponíveis, foi realizado o treino de um classificador linear utilizado o algoritmo de otimização SGD. Para esse classificador, segmentou-se a imagem em regiões de 20 por 20 pixels, extraindo as características de médias das componentes e desvio padrão dos segmentos. Para que fosse possível realizar a classificação com um número reduzido de classes, segmentou-se os níveis de florestamento em 10 índices, onde o índice i corresponde ao ao intervalo $[10i, 10i + 10)$, para $i \in \{0, 1, ..., 8\}$, e o índice 9 corresponde ao intervalo $[90, 100]$. O erro de classificação foi de $41,0\%$ para esse classificador. A figura 4.6 exibe a predição realizada sobre um subconjunto do conjunto designado para teste.

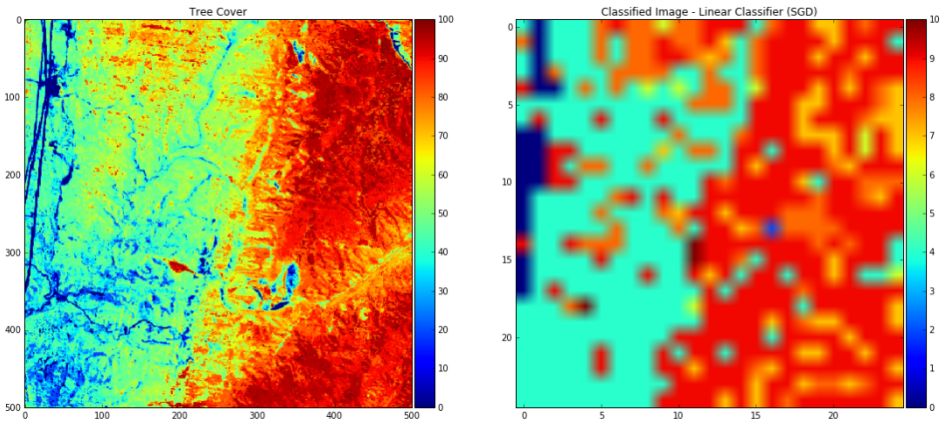


FIG. 4.6: Classificação do nível de florestamento utilizando um estimador linear

Um outro classificador foi implementado, utilizando uma rede convolucional com 18 camadas e como entradas as imagens de regiões de 20 por 20 pixels. Desta vez obteve-se um erro de $27,4\%$ no conjunto de teste. A figura 4.7 exibe a predição realizada sobre o mesmo subconjunto da imagem 4.6.

O código completo utilizado na prova de conceito pode ser consultado no Apêndice 1 do presente trabalho.

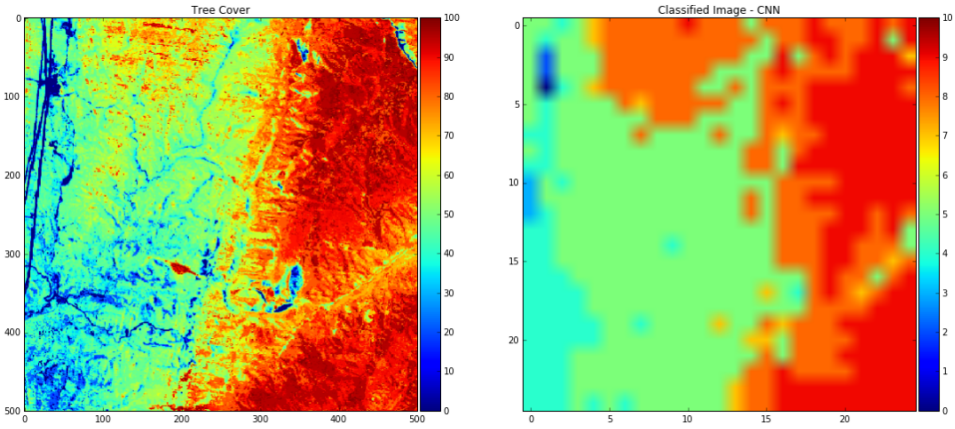


FIG. 4.7: Classificação do nível de florestamento utilizando uma rede neural convolucional

4.3 PROTÓTIPO

Após a validação realizada na prova de conceito, desenvolveu-se a plataforma web para gerência, treino e visualização de modelos que foi nomeada de *Deep Forest*. A plataforma implementa as funcionalidades previstas no modelo conceitual e foi disponibilizada na infra-estrutura da rede do Instituto Militar de Engenharia.

A figura 4.8 mostra a *landing page* do protótipo que informa ao usuário as características e funcionalidades disponíveis na plataforma. As figuras 4.9, 4.10, 4.11 e 4.12 exibem

as interfaces do protótipo implementado.

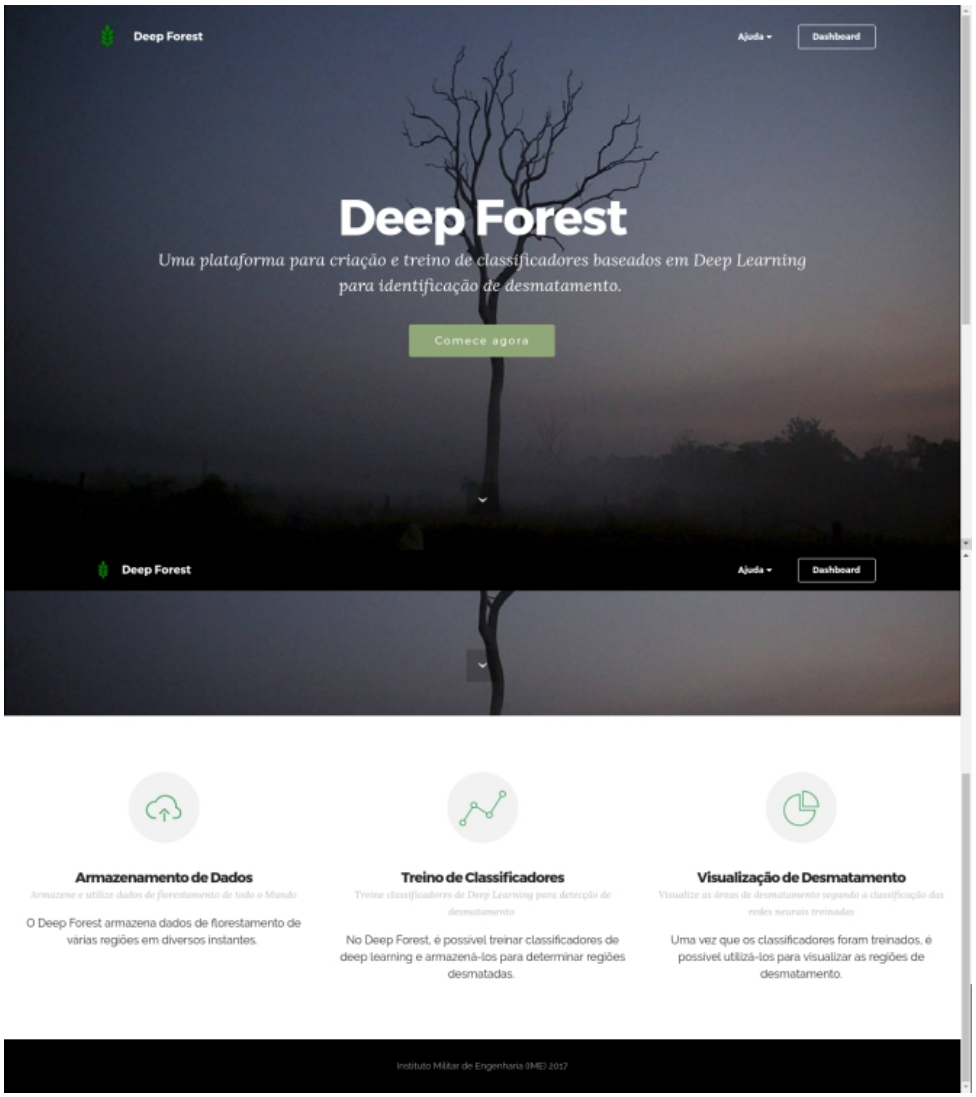


FIG. 4.8: Tela inicial do protótipo

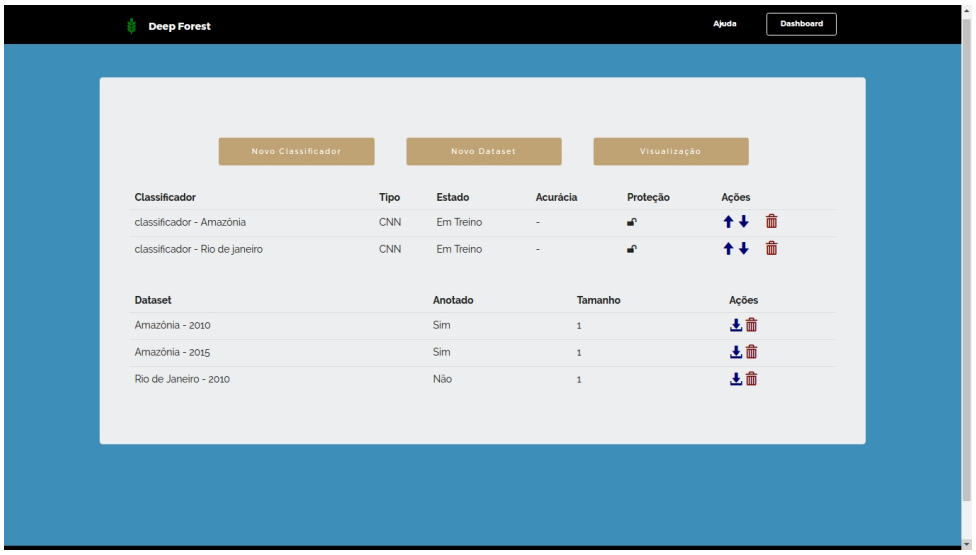


FIG. 4.9: *Dashboard* da plataforma - Permite a manipulação dos classificadores e *datasets*

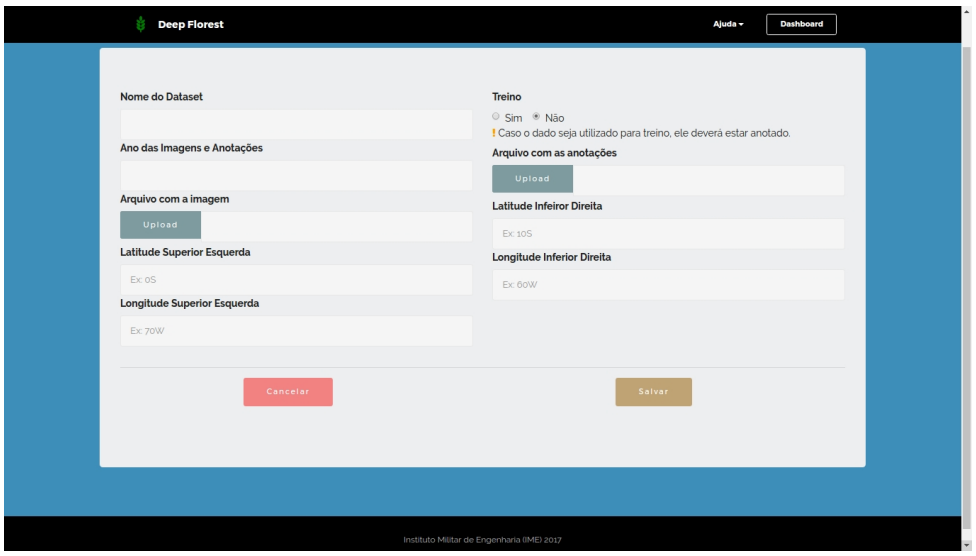


FIG. 4.10: Tela de cadastro do *dataset*

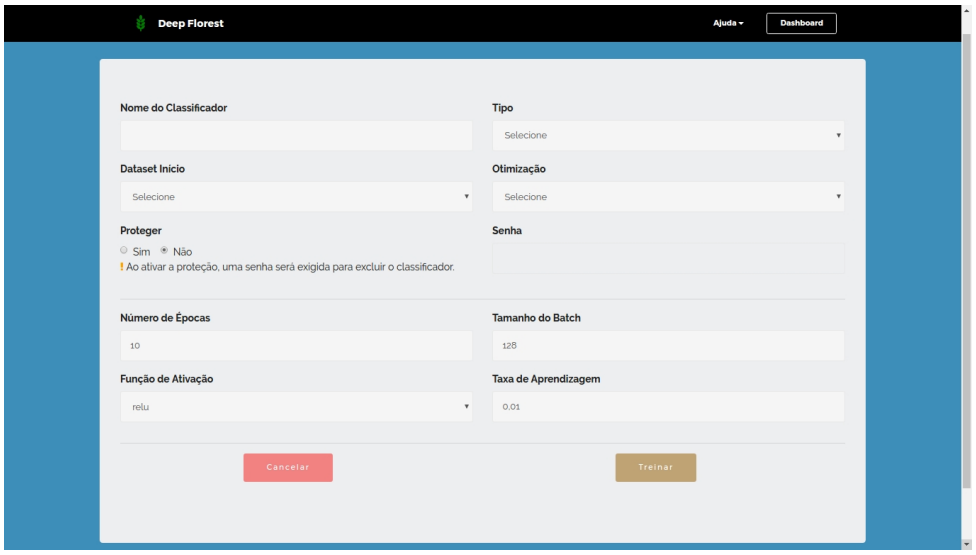


FIG. 4.11: Tela de cadastro do classificador

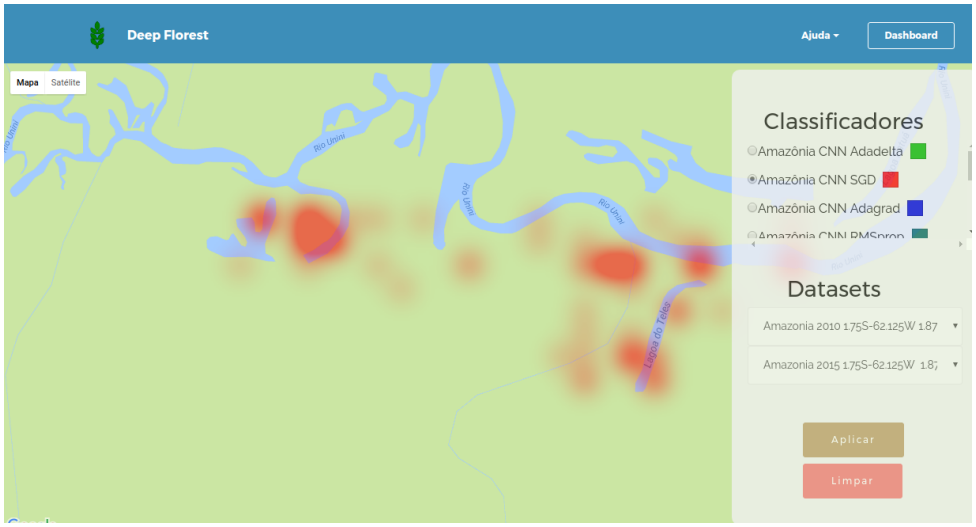


FIG. 4.12: Tela de visualização - Permite a visualização interativa da predição de desmatamento dos classificadores

5 EXPERIMENTOS E RESULTADOS

O desenvolvimento da plataforma permitiu a construção de uma bateria de experimentos de classificadores de nível de florestamento, variando os algoritmos de classificação e métodos de otimização.

A realização desses experimentos teve por objetivo comprovar a eficiência da rede neural convolucional no tratamento de problemas de visão computacional em comparação às redes neurais convencionais. Os experimentos foram executados na máquina disponibilizada pelo Laboratório de Defesa Cibernética do IME.

Para a execução dos experimentos (treinamento e cálculo da acurácia dos classificadores), foram utilizados dados de florestamento da Amazônia do ano de 2000 oriundos do trabalho de Hansen et al. (2013), em que foram anotados pelos pesquisadores envolvidos os níveis de florestamento para diversas regiões do Mundo. Mais especificamente, os experimentos utilizaram um *dataset* composto por dados de satélite da classe Landsat 7 ETM+, oriundos da região Amazônica situada no interior do território delimitado pelos pontos de coordenada geográfica (ON,70W), (0N, 60W), (10S, 70W) e (10S, 60W). Os dados de imagem foram medidos tipicamente no ano de 2000 ou no ano mais próximo dentro do intervalo 1999-2002, de tal forma que observações livres de nuvem pudessem ser tomadas. Além disso, o conjunto de imagens foi construído através das observações medianas das bandas espectrais Landsat 3, 4, 5 e 7. A reflectância normalizada de topo de atmosfera (ρ) foi transformada para um valor de 8 bits utilizando um fator de escala (g), conforme a equação abaixo:

$$DN = \rho g + 1 \tag{5.1}$$

onde DN refere-se a reflectância transformada. Os valores de g foram escolhidos independentemente para cada banda para se preservar o limite dinâmico específico de cada uma, de acordo com a tabela 5.1.

TAB. 5.1: Valores do fator de escala de reflectância para cada banda

Banda Landsat	g
Banda 3 (vermelho)	508
Banda 4 (NIR)	254
Banda 5 (SWIR)	363
Banda 7 (SWIR)	423

Os dados de florestamento propriamente ditos (*treecover*) representam o nível de

cobertura florestal presente em uma dada região, e são definidos como a porcentagem de área florestada, para cada pixel da imagem, considerando vegetação maior do que 5m de altura. Para o presente experimento, foram utilizados os dados de florestamento também do ano de 2000 referentes à mesma região representada pelos dados de imagem de satélite citados acima.

Em suma, cada pixel da imagem utilizada no experimento representa um área de 30x30m e possui um valor de florestamento correspondente que varia de 0-100% assinalando o índice de cobertura florestal observado no local no ano de 2000.

No processo de aprendizagem, foram designados 50% dos dados para treino e 50% para teste, de forma aleatória. A métrica utilizada para mensurar o ajuste dos classificadores foi a acurácia, que é usualmente utilizada em processos de aprendizagem de classificação.

A tabela 5.2 exibe os resultados obtidos nos experimentos dos classificadores.

TAB. 5.2: Classificadores

Dataset	Classificador	Otimização	Acurácia
Amazônia	CNN	Adadelta	0.8390
Amazônia	CNN	SGD	0.8597
Amazônia	CNN	Adagrad	0.8507
Amazônia	CNN	RMSprop	0.8841
Amazônia	MLP	Adadelta	0.7412
Amazônia	MLP	SGD	0.7578
Amazônia	MLP	Adagrad	0.7411
Amazônia	MLP	RMSprop	0.7444

Os classificadores que utilizavam o modelo de rede neural convolucional obtiveram acurácia superior aos que utilizaram MLP (*Multi Layer Perceptron*), que é um modelo convencional de rede neural com grande número de nós em suas camadas.

A arquitetura da rede neural convolucional utilizada é composta das seguintes camadas: camada de entrada, convolução, convolução, max-polling, convolução, convolução, max-polling, dropout, totalmente conectada, dropout e softmax.

A arquitetura da MLP utilizada é composta das seguintes camadas, na sequência: camada de entrada, totalmente conectada, dropout, totalmente conectada, softmax. Utilizou-se portanto duas camadas ocultas na arquitetura da MLP. O número de nós total nas camadas ocultas foi de 384, sendo 128 na primeira camada oculta e 256 na segunda camada oculta.

As redes foram treinadas utilizando 1 época e taxa de aprendizagem de 10^{-4}.

Apesar de ser simples a execução dos experimentos pela plataforma, pois ela abstrai todos os detalhes de infra-estrutura no processo de aprendizagem, o tempo despendido

para executar todos os experimentos foi superior a uma semana. Isso se deve ao fato de que o volume de dados utilizados no treinamento foi da ordem de Gigabytes, o que tornou a aprendizagem demorada.

O desempenho superior das CNN's era esperado, uma vez que esse tipo de modelo possui de forma inerente a capacidade de aprender e reconhecer *features* para classificação de imagens. Por essa razão, esse modelo costuma ser utilizado nos mais diversos problemas de *Machine Learning* no domínio da Visão Computacional. O método de otimização que obteve melhor desempenho foi o Adagrad nas redes neurais convolucionais e o SGD nas redes MLP.

Uma vez que os classificadores estavam treinados, foi possível a realização do cálculo de desmatamento entre os anos de 2010 e 2015. Nessa etapa, a plataforma foi utilizada para a visualização e comparação dos focos desmatamento para algumas regiões da Amazônia.

A figura 5.1 exibe uma comparação realizada entre os classificadores CNN com Adgrad e CNN com SGD, em azul e vermelho, respectivamente. Quanto maior o desmatamento, mais forte é a cor plotada sobre o mapa. Para essa região, pode-se observar maior incidência de desmatamento na proximidade dos rios, o que pode ter sido causada em razão da movimentação do curso dos rios ao decorrer dos 5 anos.

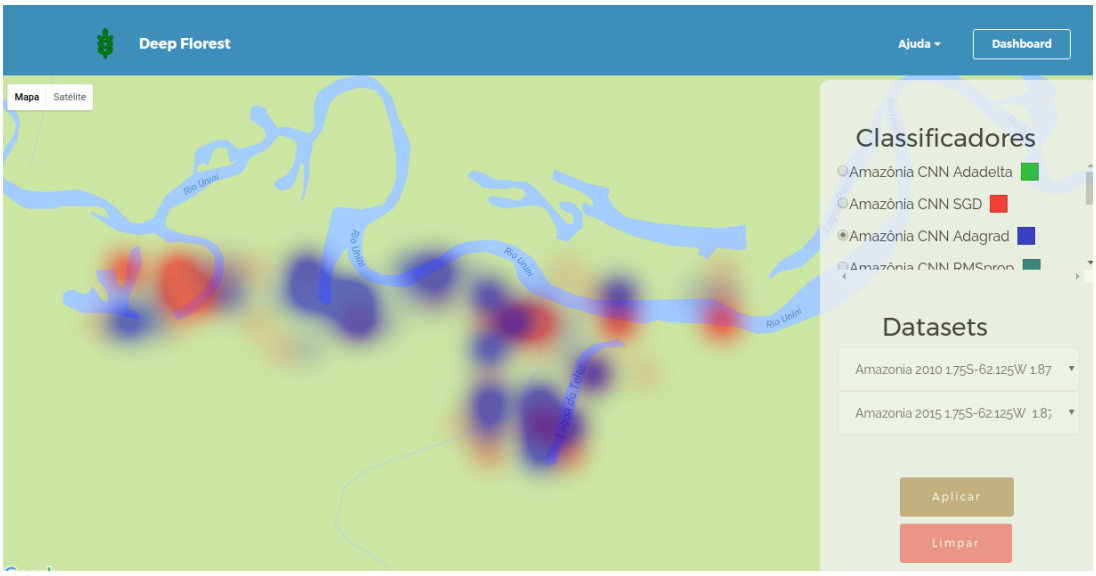


FIG. 5.1: Comparação entre as definições de desmatamento segundo dois classificadores

6 CONSIDERAÇÕES FINAIS

Pode-se afirmar que o objetivo do trabalho – o de desenvolver uma ferramenta customizável e interativa capaz de auxiliar na observação e predição das áreas de desmatamento – foi plenamente atingido. A plataforma será útil para atender a demanda crescente que existe na área de controle ambiental, podendo atuar como ferramenta de apoio a decisão em benefício aos órgãos governamentais ou ,até mesmo, às entidades privadas.

Além da plataforma em si, foram desenvolvidos testes e experimentos com a finalidade de ajustar e validar a ferramenta. Os experimentos mostraram que as Redes Neurais Convolucionais (CNN) obtiveram um desempenho consideravelmente superior comparadas em relação às redes MLP (*Multilayer Perceptron*). Com base nos classificadores gerados por esses experimentos, foi possível também visualizar de forma interativa as regiões desmatadas e observar a consistência entre todos os modelos testados na detecção das regiões de desmatamento.

Como trabalho futuro, seria interessante o ajuste da plataforma para que esta seja ainda mais customizável e possa contemplar outros tipos de classificadores e formatos de anotação para os dados de florestamento. Além disso, a possibilidade de automatizar a ferramenta para que monitore regiões de interesse de tempos em tempos também seria de grande valia para entidades de proteção ambiental.

7 CONCLUSÃO

A aprendizagem profunda e a metodologia aplicada se mostraram eficientes na predição de focos de desmatamento, uma vez que foi possível obter acurácia superior a 80% na classificação do nível de florestamento em imagens de satélite de alta resolução.

Os classificadores que obtiveram maior acurácia foram as redes neurais convolucionais, o que era esperado uma vez que esse tipo de rede neural é especializada no aprendizado *features* de imagens.

Além disso, uma plataforma aberta que permite a gerência, treino e visualização de redes neurais de forma automatizada é de grande utilidade para o combate e prevenção de focos desmatamento, uma vez que não há disponível no mercado uma ferramenta com essa especificação e esse nível de customização. Além disso, a infra-estrutura promovida pela plataforma se provou eficaz na gerência do treino de múltiplos de classificadores, variando parâmetros das redes e métodos de otimização.

8 REFERÊNCIAS BIBLIOGRÁFICAS

ABU-MOSTAFA, Y. S.; MAGDON-ISMAIL, M. ; LIN, H.-T. **Learning From Data**. [S.l.]: AMLBook, 2012. ISBN 1600490069, 9781600490064.

BURGES, C.; SHAKED, T.; RENSHAW, E.; LAZIER, A.; DEEDS, M.; HAMILTON, N. ; HULLENDER, G. Learning to rank using gradient descent. In: PROCEEDINGS OF THE 22ND INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 1., ICML '05, 1., 2005. **Anais**... New York, NY, USA: ACM, 2005, p. 89–96. Disponível em: <http://doi.acm.org/10.1145/1102351.1102363>. Acesso em: 17 abr. 2017.

CHAKRAVARTY, S.; DEY, A.; SURESH, C.; SHUKLA, G. ; GHOSH, S. **Deforestation: Causes, Effects and Control Strategies**. [S.l.]: INTECH Open Access Publisher, 2012. ISBN 9789535105695.

DENG, L.; YU, D. Deep learning: Methods and applications. **Found. Trends Signal Process.**, v. 7, p. 197–387, 2014. Disponível em: <http://dx.doi.org/10.1561/2000000039>. Acesso em: 23 abr. 2017.

DUCHI, J.; HAZAN, E. ; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. **J. Mach. Learn. Res.**, v. 12, p. 2121–2159, 2011. Disponível em: <http://dl.acm.org/citation.cfm?id=1953048.2021068>. Acesso em: 27 abr. 2017.

FACELI, K. **Inteligência Artificial. Uma Abordagem de Aprendizado de Máquina**. 1. ed. Rio de Janeiro: LTC, 2011. 394 p.

HANSEN, M. C.; POTAPOV, P. V.; MOORE, R.; HANCHER, M.; TURUBANOVA, S. A.; TYUKAVINA, A.; THAU, D.; STEHMAN, S. V.; GOETZ, S. J.; LOVELAND, T. R.; KOMMAREDDY, A.; EGOROV, A.; CHINI, L.; JUSTICE, C. O. ; TOWNSHEND, J. R. G. High-resolution global maps of 21st-century forest cover change. **Science**, v. 342, n. 6160, p. 850–853, 2013. Disponível em: <http://science.sciencemag.org/content/342/6160/850>. Acesso em: 1 maio 2017.

HANSEN, POTAPOV, MOORE, HANCHER ET AL. Global Forest Change 2000–2014. Disponível em: <https://earthenginepartners.appspot.com/science-2013-global-forest/download_v1.2.html>. Acesso em: 25 abr. 2017.

GEOFF HINTON. tricks in SGD. Disponível em: <http://www.cs.toronto.edu/t̃ijmen/csc321/slides/lecture_slides_lec6.pdf>. Acesso em: 10 out 2017.

IAN GOODFELLOW, YOSHUA BENGIO, A. C. **Deep Learning (Adaptive Computation and Machine Learning series)**. 1. ed. [S.l.]: The MIT Press, 2016. 800 p.

IMAGENET. Imagenet. Disponível em: <http://image-net.org/>. Acesso em: 24 abr. 2017.

INPE-OBT. Projeto PRODES : Monitoramento da Floresta Amazônica Brasileira por Satélite. Disponível em: <http://www.obt.inpe.br/prodes/index.php>. Acesso em: 8 maio 2017.

KRIZHEVSKY, A.; SUTSKEVER, I. ; HINTON, G. E. **ImageNet Classification with Deep Convolutional Neural Networks**. [S.l.]: Curran Associates, Inc., 2012. 1097–1105 p.

LOWE, D. G. Three-dimensional object recognition from single two-dimensional images. **Artificial Intelligence**, v. 31, n. 3, p. 355 – 395, 1987. Disponível em: <http://www.sciencedirect.com/science/article/pii/0004370287900701>. Acesso em: 29 mar. 2017.

PHYS-ORG. Remote sensing and forest inventories contribute to saving tropical forests. Disponível em: <https://phys.org/news/2016-06-remote-forest-contribute-tropical-forests.html>. Acesso em: 02 out 2017.

SCHAPIRE, R. E.; SINGER, Y. Boostexter: A boosting-based system for text categorization. **Machine Learning**, v. 39, n. 2, p. 135–168, 2000. Disponível em: <http://dx.doi.org/10.1023/A:1007649029923>. Acesso em: 10 abr. 2017.

SCHÜRMANN, J. **Pattern Classification: A Unified View of Statistical and Neural Approaches**. New Jersey: Wiley-Interscience, 1996. 392 p.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; VAN DEN DRIESSCHE, G.; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M.; DIELEMAN, S.; GREWE, D.; NHAM, J.; KALCHBRENNER, N.; SUTSKEVER, I.; LILLICRAP, T.; LEACH, M.; KAVUKCUOGLU, K.; GRAEPEL,

T. ; HASSABIS, D. Mastering the game of go with deep neural networks and tree search. **Nature**, v. 529, n. 7587, p. 484–489, 2016.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **CoRR**, v. abs/1409.1556, 2014. Disponível em: <http://arxiv.org/abs/1409.1556>. Acesso em: 1 maio 2017.

SOARES, R. A. Causas do desmatamento no brasil e seu ordenamento no contexto mundial. **Revista de Economia e Sociologia Rural**, v. 50, 2012. Disponível em: <http://dx.doi.org/10.1590/S0103-20032012000100007>. Acesso em: 28 mar. 2017.

SZEGEDY, C.; IOFFE, S. ; VANHOUCKE, V. Inception-v4, inception-resnet and the impact of residual connections on learning.. **CoRR**, v. abs/1602.07261, 2016. Disponível em: <http://dblp.uni-trier.de/db/journals/corr/corr1602.html#SzegedyIV16>. Acesso em: 25 abr. 2017.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V. ; RABINOVICH, A. Going deeper with convolutions. **CoRR**, v. abs/1409.4842, 2014. Disponível em: <http://dblp.uni-trier.de/db/journals/corr/corr1409.html#SzegedyLJSRAEVR14>. Acesso em: 25 abr. 2017.

WIKIWAND. Convolutional neural network. Disponível em: <http://www.wikiwand.com/en/Convolutional_neural_network>. Acesso em: 24 abr. 2017.

WORLD-BANK. Forest area (sq. km). Disponível em: <https://data.worldbank.org/indicator/AG.LND.FRST.K2>. Acesso em: 01 out 2017.

ZEILER, M. D. ADADELTA: an adaptive learning rate method. **CoRR**, v. abs/1212.5701, 2012. Disponível em: <http://arxiv.org/abs/1212.5701>. Acesso em: 10 out 2017.

ŠIMIĆ DE TORRES, I. **ANALYSIS OF SATELLITE IMAGES TO TRACK DEFORESTATION**. 2016. 56 f. Trabalho de Conclusão de Curso (Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona) – Universitat Politècnica de Catalunya, Barcelona, 2016.

9 APÊNDICES

Proof of Concept

May 1, 2017

In [1]:
```
# packages
import numpy as np
import tensorflow
import scipy.ndimage
import tifffile
import os
import matplotlib.pyplot as plt
import PIL
from mpl_toolkits.axes_grid1 import make_axes_locatable
from IPython.display import clear_output
% matplotlib inline
```

In [2]:
```
# constants
image_width = 40000
# It will separate the original image in 6400 images 500 x 500
separate_width = 500
length_classification = 20
total_images =  image_width / separate_width

# custom flags
show_image = False
cut_and_save_images = False

# custom path and coordinate
data_path = "C:/Users/IBM_ADMIN/workspace/pfc"
coordinates = {"lat": "20S", "lon": "070W"}
dir_name = coordinates["lat"] + coordinates["lon"]

# file paths
tree_cover_path = "{0}/Hansen_GFC2015_treecover2000_{1}_{2}.tif".\
    format(data_path, coordinates["lat"], coordinates["lon"])
first_path = "{0}/Hansen_GFC2015_first_{1}_{2}.tif".\
    format(data_path, coordinates["lat"], coordinates["lon"])
last_path = "{0}/Hansen_GFC2015_last_{1}_{2}.tif".\
    format(data_path, coordinates["lat"], coordinates["lon"])
loss_path = "{0}/Hansen_GFC2015_path_{1}_{2}.tif".\
    format(data_path, coordinates["lat"], coordinates["lon"])
```

```
In [3]: def cut_and_save(directory, name, matrix, length):
            try:
                os.makedirs("{0}/numpy_files/".\
                            format(data_path))
            except FileExistsError:
                pass
            try:
                os.makedirs("{0}/numpy_files/{1}".\
                            format(data_path, directory))
            except FileExistsError:
                pass

            limit = int(len(matrix) / length)
            for i in range(limit):
                for j in range(limit):
                    sub_matrix = matrix[i * length : i * length + length,
                                        j * length : j * length + length]
                    np.save("{0}/numpy_files/{1}/{2}-{3}-{4}.npy".\
                            format(data_path, directory, name, i, j),
                                sub_matrix)

In [4]: def separate_matrix(matrix, length):
            list_matrix = list()
            limit = int(len(matrix) / length)
            for i in range(limit):
                for j in range(limit):
                    sub_matrix = matrix[i * length : i * length + length,
                                        j * length : j * length + length]
                    list_matrix.append(sub_matrix)
            return list_matrix

In [5]: def open_images(directory, name, total, length,
                        percent=[0,100], label=False):
            list_matrix = list()
            limit_min = int(total * percent[0] * 1.0 / 100)
            limit_max = int(total * percent[1] * 1.0 / 100)
            for i in range(limit_min, limit_max):
                for j in range(total):
                    sub_matrix = \
                        np.load("{0}/numpy_files/{1}/{2}-{3}-{4}.npy".\
                                format(data_path, directory, name, i, j))
                    if not label:
                        list_matrix += separate_matrix(sub_matrix, length)
                    else:
                        separate_current_list = \
                            separate_matrix(sub_matrix, length)
                        list_matrix.\
                            append([int((np.mean(current_matrix) + 1) / 10)\
```

```
                                        for current_matrix \
                                            in separate_current_list])
    return np.array(list_matrix)


def return_feature(directory, name, length, i, j):
    list_features = list()
    sub_matrix = np.load("{0}/numpy_files/{1}/{2}-{3}-{4}.npy".\
                         format(data_path, directory, name, i, j))
    limit = int(len(sub_matrix) / length)
    for u in range(limit):
        for v in range(limit):
            sub_matrix_current =\
                sub_matrix[u * length : u * length + length,
                           v * length : v * length + length]
            mean_colors = sub_matrix_current.mean(0).mean(0)
            var_colors = sub_matrix_current.var(0).var(0)
            list_features.append(mean_colors.tolist()
                                 + var_colors.tolist())
    return list_features


def open_features(directory, name, total, length):
    list_features = list()
    for i in range(total):
        for j in range(total):
            list_features += return_feature(directory,
                                            name, length, i, j)
    return list_features


def classification_criteria(directory, name, length, i, j):
    list_classification = list()
    sub_matrix = np.load("{0}/numpy_files/{1}/{2}-{3}-{4}.npy".\
                         format(data_path, directory, name, i, j))
    limit = int(len(sub_matrix) / length)
    for u in range(limit):
        for v in range(limit):
            sub_matrix_current = \
                sub_matrix[u * length : u * length + length,
                           v * length : v * length + length]
            list_classification.\
                append(int((np.mean(sub_matrix_current) + 1) / 10))
    return list_classification


def open_classifications(directory, name, total, length):
    list_classification = list()
```

```
            for i in range(total):
                for j in range(total):
                    list_classification += \
                        classification_criteria(directory,\
                                                name, length, i, j)
            return list_classification
```

```
In [6]: # open, cut and save the first image:
        if cut_and_save_images:
            first_image = tifffile.imread(first_path)
            cut_and_save(dir_name, "first", first_image, separate_width)
            first_image = np.NaN # release memory
```

```
In [7]: # open, cut and save the tree cover:
        if cut_and_save_images:
            treee_cover_image = tifffile.imread(tree_cover_path)
            cut_and_save(dir_name, "treecover",
                         treee_cover_image, separate_width)
            treee_cover_image = np.NaN # release memory
```

```
In [8]: features_example = return_feature(dir_name, "first",
                                           length_classification, 20, 20)
        print(("Features:\n-----------------\nMean 1 = {0}\n" +
               "Mean 2 = {1}\nMean 3 = {2}\nMean 4 = {3}").\
               format(features_example[0][0], features_example[0][1],
                      features_example[0][2], features_example[0][3]))
```

```
Features:
-----------------
Mean 1 = 151.6625
Mean 2 = 93.99999999999999
Mean 3 = 166.3075
Mean 4 = 179.14999999999998
```

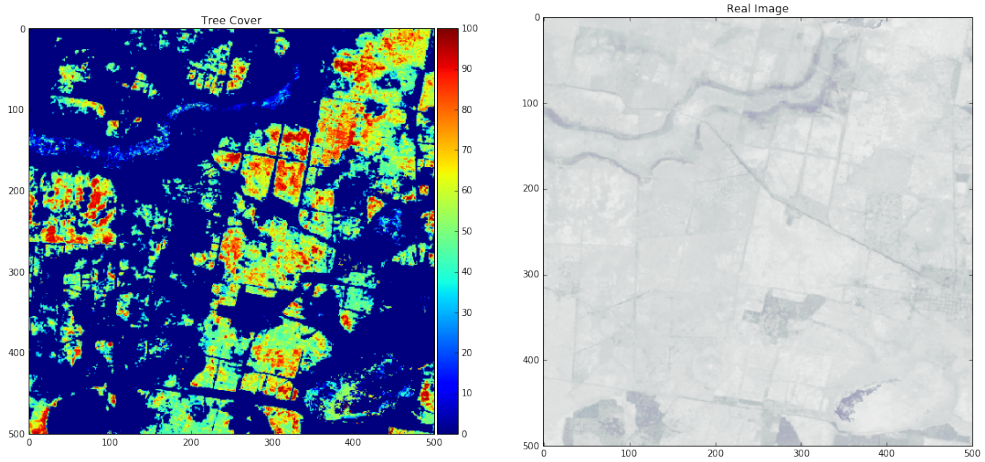
```
In [9]: # tree cover anotation in cell (40, 40)
        fig, (ax1, ax2) = plt.subplots(1, 2)
        fig.set_size_inches(18.5, 10.5)

        tree_cover_example = np.load(("{0}/numpy_files/{1}/" +
                                      "{2}-{3}-{4}.npy").\
                                      format(data_path, dir_name,
                                             "treecover", 76, 76))
        im1 = ax1.imshow(PIL.Image.fromarray(tree_cover_example))
        divider1 = make_axes_locatable(ax1)
        cax1 = divider1.append_axes("right", size="5%", pad=0.05)
        cbar1 = plt.colorbar(im1, cax=cax1)
        ax1.set_title("Tree Cover")
```

```
        # image cell (40, 40)
        image_example = np.load("{0}/numpy_files/{1}/{2}-{3}-{4}.npy".\
                                format(data_path, dir_name, "first", 76, 76))
        w, h, d = image_example.shape
        pil_image = PIL.Image.frombytes("RGBA", (w, h),
                                        image_example.tostring())

        im2 = ax2.imshow(pil_image)
        ax2.set_title("Real Image")
```

```
Out[9]: <matplotlib.text.Text at 0xa1ec4e0>
```




```
In [10]: # get features from all sub-images
         all_features = open_features(dir_name, "first",
                                      int(total_images),
                                      length_classification)

         # get the labels from all sub-images
         all_labels = open_classifications(dir_name,
                                           "treecover",
                                           int(total_images),
                                           length_classification)
```

```
In [11]: # separating the data in train and test
         train_data_x = \
             all_features[0 : int(len(all_features) / 3)]
         train_data_y = \
             all_labels[0 : int(len(all_labels) / 3)]
         test_data_x = \
             all_features[int(len(all_features) / 3) \
```

```
                    : int(len(all_features))]
        test_data_y = \
            all_labels[int(len(all_features) / 3)  \
                       : int(len(all_labels))]
```

```
In [12]: def success(predicted, result):
             compare = predicted == result
             count_true = 0
             count_false = 0
             for equal in compare:
                 if equal:
                     count_true += 1
                 else:
                     count_false += 1
             return count_true / (count_true + count_false)


         def compose_matrix(class_list, image_width,
                            separate_width, length_classification):
             limit = int(image_width / length_classification)
             matrix = list()
             for i in range(limit):
                 matrix.append(limit * [-1])
             row_block_length = int(separate_width / length_classification)
             num_block_row = int(image_width / separate_width)
             for i in range(len(class_list)):
                 block_num = int(i / (row_block_length * row_block_length))
                 u = block_num // num_block_row
                 v = block_num % num_block_row
                 r = i % (row_block_length * row_block_length)
                 x = u * row_block_length + r / (row_block_length)
                 y = v * row_block_length + r % (row_block_length)
                 matrix[int(x)][int(y)] = class_list[i]
             return matrix
```

```
In [13]: # linear classifier - SGD
         from sklearn.linear_model import SGDClassifier
         clf = SGDClassifier(loss="hinge", penalty="l2",
                             n_iter=15, n_jobs=1)
         clf.fit(train_data_x, train_data_y)
         print("Success = {0}".format(success(clf.predict(test_data_x),
                                              test_data_y)))
```

```
Success = 0.5815716773035403
```

```
In [14]: # apply classifier in all data
         class_list = clf.predict(all_features)
         classified_image = compose_matrix(class_list, image_width,
```

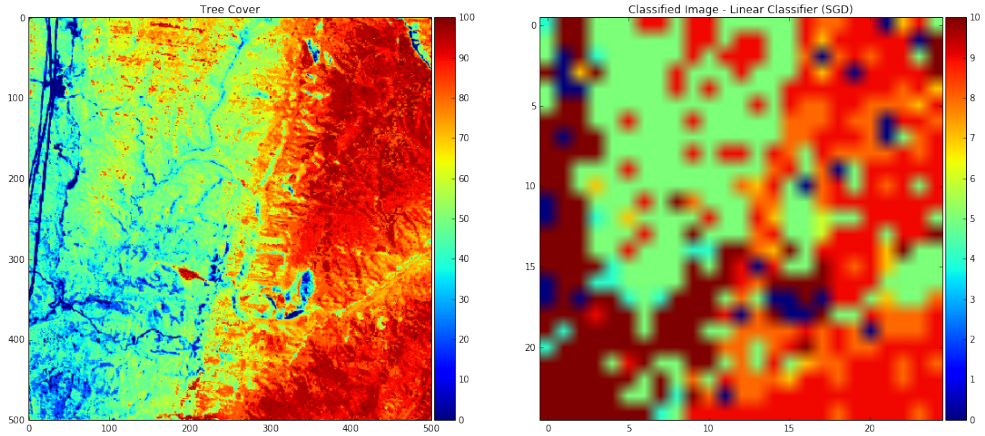
```
                                                separate_width,
                                                length_classification)
```

In [15]:
```
# compare treecover and classfied data
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_size_inches(18.5, 10.5)

tree_cover_example = \
    np.load("{0}/numpy_files/{1}/{2}-{3}-{4}.npy".\
            format(data_path, dir_name, "treecover", 40, 40))
im1 = ax1.imshow(PIL.Image.fromarray(tree_cover_example))
divider1 = make_axes_locatable(ax1)
cax1 = divider1.append_axes("right", size="5%", pad=0.05)
cbar1 = plt.colorbar(im1, cax=cax1)
ax1.set_title("Tree Cover")

# image cell (40, 40)
limit = int(separate_width / length_classification)
class_matrix = list()
for i in range(limit):
    new_row = list()
    for j in range(limit):
        new_row.append(classified_image[40 * limit + i]
                       [40 * limit  + j])
    class_matrix.append(new_row)
im2 = ax2.imshow(class_matrix, vmin=0, vmax=10)
divider2 = make_axes_locatable(ax2)
cax2 = divider2.append_axes("right", size="5%", pad=0.05)
cbar2 = plt.colorbar(im2, cax=cax2)
ax2.set_title("Classified Image - Linear Classifier (SGD)")
```

Out[15]:
```
<matplotlib.text.Text at 0x6fbe7278>
```



```
In [16]: open_labels = True # set True if it is not opened
         percent_train_min = 41
         percent_train_max = 57
         percent_test_min = 53
         percent_test_max = 61

         # Release Memory
         all_features = None
         train_data_x = None
         train_data_y = None
         test_data_x = None
         test_data_y = None
         class_list = None
         classified_image = None
         all_images = None
         all_labels = None

         # Open Images
         train_data_x = open_images(dir_name,
                                    "first",
                                    int(total_images),
                                    length_classification,
                                    percent=[percent_train_min,
                                             percent_train_max])

         test_data_x = open_images(dir_name,
                                   "first",
                                   int(total_images),
                                   length_classification,
                                   percent=[percent_test_min,
                                            percent_test_max])

         train_data_y = open_images(dir_name,
                                    "treecover",
                                    int(total_images),
                                    length_classification,
                                    percent=[percent_train_min,
                                             percent_train_max],
                                    label=True)

         test_data_y = open_images(dir_name,
                                   "treecover",
                                   int(total_images),
                                   length_classification,
                                   percent=[percent_test_min,
                                            percent_test_max],
                                   label=True)
```

In [17]:
```
'''Trains a simple convnet on the MNIST dataset.
Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tuning).
16 seconds per epoch on a GRID K520 GPU.
'''
# This is only an example
run_example = False

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

if run_example:
    batch_size = 128
    num_classes = 10
    epochs = 12

    # input image dimensions
    img_rows, img_cols = 28, 28

    # the data, shuffled and split between train and test sets
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

    if K.image_data_format() == 'channels_first':
        x_train = x_train.reshape(x_train.shape[0],
                                  1, img_rows, img_cols)
        x_test = x_test.reshape(x_test.shape[0],
                                1, img_rows, img_cols)
        input_shape = (1, img_rows, img_cols)
    else:
        x_train = x_train.reshape(x_train.shape[0],
                                  img_rows, img_cols, 1)
        x_test = x_test.reshape(x_test.shape[0],
                                img_rows, img_cols, 1)
        input_shape = (img_rows, img_cols, 1)

    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255
    x_test /= 255
    print('x_train shape:', x_train.shape)
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')
```

```
            # convert class vectors to binary class matrices
            y_train = keras.utils.to_categorical(y_train, num_classes)
            y_test = keras.utils.to_categorical(y_test, num_classes)

            model = Sequential()
            model.add(Conv2D(32, kernel_size=(3, 3),
                             activation='relu',
                             input_shape=input_shape))
            model.add(Conv2D(64, (3, 3), activation='relu'))
            model.add(MaxPooling2D(pool_size=(2, 2)))
            model.add(Dropout(0.25))
            model.add(Flatten())
            model.add(Dense(128, activation='relu'))
            model.add(Dropout(0.5))
            model.add(Dense(num_classes, activation='softmax'))

            model.compile(loss=keras.losses.categorical_crossentropy,
                          optimizer=keras.optimizers.Adadelta(),
                          metrics=['accuracy'])

            model.fit(x_train, y_train,
                      batch_size=batch_size,
                      epochs=epochs,
                      verbose=1,
                      validation_data=(x_test, y_test))
            score = model.evaluate(x_test, y_test, verbose=0)
            print('Test loss:', score[0])
            print('Test accuracy:', score[1])
```

```
Using TensorFlow backend.
```

```
In [18]: batch_size = 32
         num_classes = 11
         epochs = 1
         data_augmentation = True

         # The data, shuffled and split between train and test sets:
         print('x_train shape:', train_data_x.shape)
         print(train_data_x.shape[0], 'train samples')
         print(train_data_x.shape[0], 'test samples')

         # Convert class vectors to binary class matrices.
         train_data_y = keras.utils.\
             to_categorical(train_data_y, num_classes)
         test_data_y = keras.utils.\
             to_categorical(test_data_y, num_classes)
```

```
model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=train_data_x.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

if not data_augmentation:
    print('Not using data augmentation.')
    model.fit(x_train, train_data_y,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(train_data_x, test_data_y),
              shuffle=True)
else:
    print('Using real-time data augmentation.')
    # This will do preprocessing and realtime data augmentation:
    datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
```

```
                zca_whitening=False,
                rotation_range=0,
                width_shift_range=0.1,
                height_shift_range=0.1,
                horizontal_flip=True,
                vertical_flip=False)

            # Compute quantities required for feature-wise normalization
            datagen.fit(train_data_x)

            # Fit the model on the batches generated by datagen.flow().
            model.fit_generator\
                (datagen.flow(train_data_x, train_data_y,
                              batch_size=batch_size),
                 steps_per_epoch=train_data_x.shape[0] // batch_size,
                 epochs=epochs,
                 validation_data=(test_data_x, test_data_y))
        clear_output()
```

```
In [19]: score = model.evaluate(test_data_x, test_data_y, verbose=0)
         print('Test loss:', score[0])
         print('Test accuracy:', score[1])
```

```
Test loss: 0.714100446508
Test accuracy: 0.71621
```

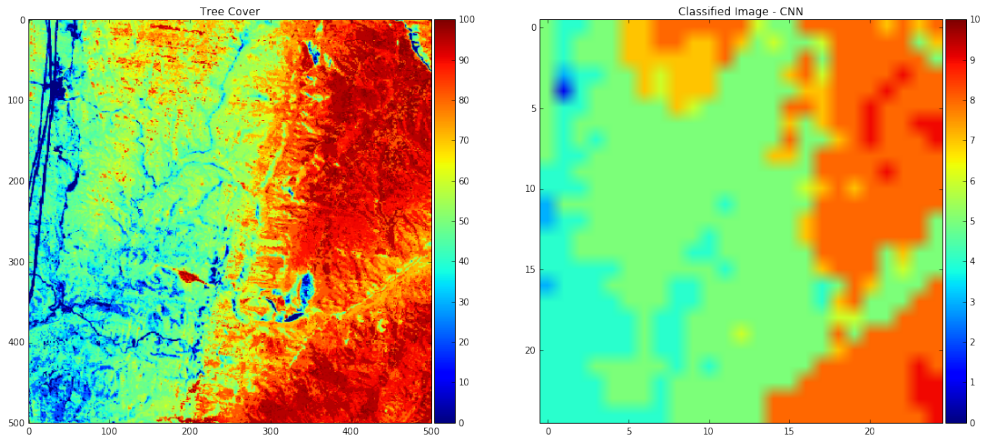
```
In [20]: def get_class_from_list(list_class):
             index = -1
             max_prob = -1
             for i in range(len(list_class)):
                 if list_class[i] > max_prob:
                     index = i
                     max_prob = list_class[i]
             return index
```

```
In [21]: # compare treecover and classfied data
         fig, (ax1, ax2) = plt.subplots(1, 2)
         fig.set_size_inches(18.5, 10.5)

         tree_cover_example = \
             np.load("{0}/numpy_files/{1}/{2}-{3}-{4}.npy".\
                     format(data_path, dir_name, "treecover", 40, 40))
         im1 = ax1.imshow(PIL.Image.fromarray(tree_cover_example))
         divider1 = make_axes_locatable(ax1)
         cax1 = divider1.append_axes("right", size="5%", pad=0.05)
         cbar1 = plt.colorbar(im1, cax=cax1)
         ax1.set_title("Tree Cover")
```

```
limit = int(separate_width / length_classification)
image_cell = \
    np.load("{0}/numpy_files/{1}/{2}-{3}-{4}.npy".\
            format(data_path, dir_name, "first", 40, 40))
list_images = separate_matrix(image_cell, length_classification)
list_predicted = \
    [get_class_from_list(model.predict(np.array([image]))[0])\
     for image in list_images]
class_matrix = list()
for i in range(limit):
    new_row = list()
    for j in range(limit):
        new_row.append(list_predicted[i * limit + j])
    class_matrix.append(new_row)
im2 = ax2.imshow(class_matrix, vmin=0, vmax=10)
divider2 = make_axes_locatable(ax2)
cax2 = divider2.append_axes("right", size="5%", pad=0.05)
cbar2 = plt.colorbar(im2, cax=cax2)
ax2.set_title("Classified Image - CNN")
```

Out[21]: <matplotlib.text.Text at 0xca0d978>



In [22]: model.summary()

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 20, 20, 32)	1184
activation_1 (Activation)	(None, 20, 20, 32)	0

```
conv2d_2 (Conv2D)            (None, 18, 18, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 18, 18, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 9, 9, 32)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 9, 9, 32)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 9, 9, 64)          18496
_________________________________________________________________
activation_3 (Activation)    (None, 9, 9, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 64)          36928
_________________________________________________________________
activation_4 (Activation)    (None, 7, 7, 64)          0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 3, 3, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               295424
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 11)                5643
_________________________________________________________________
activation_6 (Activation)    (None, 11)                0
=================================================================
Total params: 366,923.0
Trainable params: 366,923.0
Non-trainable params: 0.0
_________________________________________________________________
```