Redes Neurais Convolucionais de Profundidade para Reconhecimento de Textos em Imagens de CAPTCHA

Vitor Arins Pinto[1]

[1]Departamento de Informática e Estatística - Universidade Federal de Santa Catarina (UFSC)
Santa Catarina – SC – Brazil

vitor.arins@grad.ufsc.br

Abstract. *Currently many applications on the Internet follow the policy of keeping some data accessible to the public. In order to do this, it's necessary to develop a portal that is robust enough to ensure that all people can access this data. But the requests made to recover public data may not always come from a human. Companies specializing in Big data have a great interest in data from public sources in order to make analysis and forecasts from current data. With this interest, Web Crawlers are implemented. They are responsible for querying data sources thousands of times a day, making several requests to a website. This website may not be prepared for such a great volume of inquiries in a short period of time. In order to prevent queries to be made by computer programs, institutions that keep public data invest in tools called CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart). These tools usually deal with images containing text and the user must enter what he or she sees in the image. The objective of the proposed work is to perform the text recognition in CAPTCHA images through the application of convolutional neural networks.*

Resumo. *Atualmente, muitas aplicações na Internet seguem a política de manter alguns dados acessíveis ao público. Para isso é necessário desenvolver um portal que seja robusto o suficiente para garantir que todas as pessoas possam acessá-lo. Porém, as requisições feitas para recuperar dados públicos nem sempre vêm de um ser humano. Empresas especializadas em Big data possuem um grande interesse em fontes de dados públicos para poder fazer análises e previsões a partir de dados atuais. Com esse interesse, Web Crawlers são implementados. Eles são responsáveis por consultar fontes de dados milhares de vezes ao dia, fazendo diversas requisições a um website. Tal website pode não estar preparado para um volume de consultas tão grande em um período tão curto de tempo. Com o intuito de impedir que sejam feitas consultas por programas de computador, as instituições que mantêm dados públicos investem em ferramentas chamadas CAPTCHA (teste de Turing público completamente automatizado, para diferenciação entre computadores e humanos). Essas ferramentas geralmente se tratam de imagens contendo um texto qualquer e o usuário deve digitar o que vê na imagem. O objetivo do trabalho proposto é realizar o reconhecimento de texto em imagens de CAPTCHA através da aplicação de redes neurais convolucionais.*

Introdução

Com o aumento constante na quantidade de informações geradas e computadas atualmente, percebe-se o surgimento de uma necessidade de tornar alguns tipos de dados acessíveis a um público maior. A fim de gerar conhecimento, muitas instituições desenvolvem portais de acesso para consulta de dados relevantes a cada pessoa. Esses portais, em forma de aplicações na Internet, precisam estar preparados para receber diversas requisições e em diferentes volumes ao longo do tempo.

Devido a popularização de ferramentas e aplicações especializadas em Big data, empresas de tecnologia demonstram interesse em recuperar grandes volumes de dados de diferentes fontes públicas. Para a captura de tais dados, *Web crawlers* são geralmente implementados para a realização de várias consultas em aplicações que disponibilizam dados públicos.

Para tentar manter a integridade da aplicação, as organizações que possuem estas informações requisitadas investem em ferramentas chamadas CAPTCHA (teste de Turing público completamente automatizado para diferenciação entre computadores e humanos). Essas ferramentas frequentemente se tratam de imagens contendo um texto qualquer e o usuário precisa digitar o que vê na imagem.

Objetivo

O objetivo geral do artigo é analisar o treinamento e aplicação de redes neurais convolucionais de profundidade para o reconhecimento de texto em imagens de CAPTCHA. Com isso será retratada a ineficiência de algumas ferramentas de CAPTCHA, mostrando como redes neurais convolucionais podem ser aplicadas em imagens a fim de reconhecer o texto contido nestas imagens.

Conceitos teóricos

Classificador Logístico

Um classificador logístico (geralmente chamado de regressão logística[Bengio and Courville 2016]) recebe como entrada uma informação, como por exemplo os pixels de uma imagem, e aplica uma função linear a eles para gerar suas predições. Uma função linear é apenas uma grande multiplicação de matriz. Recebe todas as entradas como um grande vetor que será chamado de "X", e multiplica os valores desse vetor com uma matriz para gerar as predições. Cada predição é como uma **pontuação**, que possui o valor que indica o quanto as entradas se encaixam em uma classe de saída.

$$WX + b = Y \tag{1}$$

Na equação 1, "X" é como chamaremos o vetor das entradas, "W" serão pesos e o termo tendencioso (*bias*) será representado por "b". "Y" corresponde ao vetor de pontuação para cada classe. Os pesos da matriz e o *bias* é onde age o aprendizado de máquina, ou seja, é necessário tentar encontrar valores para os pesos e para o *bias* que terão uma boa performance em fazer predições para as entradas.

Função *Softmax*

Como cada imagem pode ter um e somente um rótulo possível, é necessário transformar as pontuações geradas pelo classificador logístico em probabilidades. É essencial que a probabilidade de ser a classe correta seja muito perto de **1.0** e a probabilidade para todas as outras classes fique perto de **0.0**. Para transformar essas pontuações em probabilidades utiliza-se uma função chamada ***Softmax***[Bengio and Courville 2016].

One-Hot Encoding

Para facilitar o treinamento é preciso representar de forma matemática os rótulos de cada exemplo que iremos alimentar à rede neural. Cada rótulo será representado por um vetor de tamanho igual ao número de classes possíveis, assim como o vetor de probabilidades. No caso dos rótulos, será atribuído o valor de **1.0** para a posição referente a classe correta daquele exemplo e **0.0** para todas as outras posições. Essa tarefa é simples e geralmente chamada de *One-Hot Encoding*. Com isso é possível medir a eficiência do treinamento apenas comparando dois vetores.

Camada convolucional

A camada de uma rede neural convolucional é uma rede que compartilha os seus parâmetros por toda camada. No caso de imagens, cada exemplo possui uma largura, uma altura e uma profundidade que é representada pelos canais de cor (vermelho, verde e azul). Uma convolução consiste em coletar um trecho da imagem de exemplo e aplicar uma pequena rede neural que teria uma quantidade qualquer de saídas (K). Isso é feito deslizando essa pequena rede neural pela imagem sem alterar os pesos e montando as saídas verticalmente em uma coluna de profundidade K. No final será montada uma nova imagem de largura, altura e profundidade diferente. Essa imagem é um conjunto de **mapas de características** da imagem original. Como exemplo, transforma-se 3 mapas de carcterísticas (canais de cores) para uma quantidade K de mapas de características.

Uma rede convolucional[Dumoulin et al. 2016] será basicamente uma rede neural de profundidade. Ao invés de empilhar camadas de multiplicação de matrizes, empilha-se convoluções. No começo haverá uma imagem grande que possui apenas os valores de pixel como informação. Em seguida são aplicadas convoluções que irão "espremer" as dimensões espaciais e aumentar a profundidade. No final é possivel conectar o classificador e ainda lidar apenas com parâmetros que mapeiam o conteúdo da imagem.

ReLU

Modelos lineares são simples e estáveis numericamente, mas podem se tornar ineficientes ao longo do tempo. Portanto, para adicionar mais camadas ao modelo será necessário introduzir alguns cálculos não lineares entre camadas. Em arquiteturas de profundidade, as funções de ativação dos neurônios se chamam *Rectified Linear Units* (ReLUs)[Bengio and Courville 2016], e são capazes de introduzir os cálculos necessários aos modelos que possuem mais de uma camada. Essas são as funções não lineares mais simples que existem. Elas são lineares ($y = x$) se **x** é maior que **0**, senão ficam iguais a **0** ($y = 0$). Isso simplifica o uso de *backpropagation* e evita problemas de saturação, fazendo o aprendizado ficar muito mais rápido.

Max pooling

Após a camada convolucional adiciona-se uma camada de *pooling* que irá receber todas as convoluções e combiná-las da seguinte forma[Dumoulin et al. 2016]. Para cada ponto nos mapas de características a execução desta camada olha para uma pequena vizinhança ao redor deste ponto. Com esses valores em mãos é possível calcular o valor máximo dessa vizinhança.

Dropout

Uma forma de regularização que previne o *overfitting*[1] é o *dropout*[Krizhevsky et al.]. Supondo que temos uma camada conectada à outra em uma rede neural, os valores que vão de uma camada para a próxima podem se chamar de **ativações**. No *dropout*, são coletadas todas as ativações e aleatoriamente, para cada exemplo treinado, é atribuído o valor 0 para metade desses valores. Basicamente metade dos dados que estão fluindo pela rede neural é destruída aleatoriamente.

Camada completamente conectada

De acordo com Krizhevsky[Krizhevsky et al.], uma camada completamente conectada tem conexões com todas as ativações das camadas anteriores, assim como em redes neurais comuns. Suas ativações podem ser calculadas através de uma multiplicação de matrizes seguida da adição do fator *bias*.

Devido a quantidade de componentes presentes na estrutura de redes neurais é aparente a complexidade quanto ao entendimento do funcionamento geral. A figura 1 tenta explicar como esses componentes se conectam e em qual sequência. Os valores são completamente fictícios e não condizem com um cálculo real.

[1] O *overfitting* ocorre quando um modelo de rede neural se encaixa muito bem em um conjunto de dados e acaba memorizando propriedades do conjunto de treinamento que não servem para o conjunto de teste.

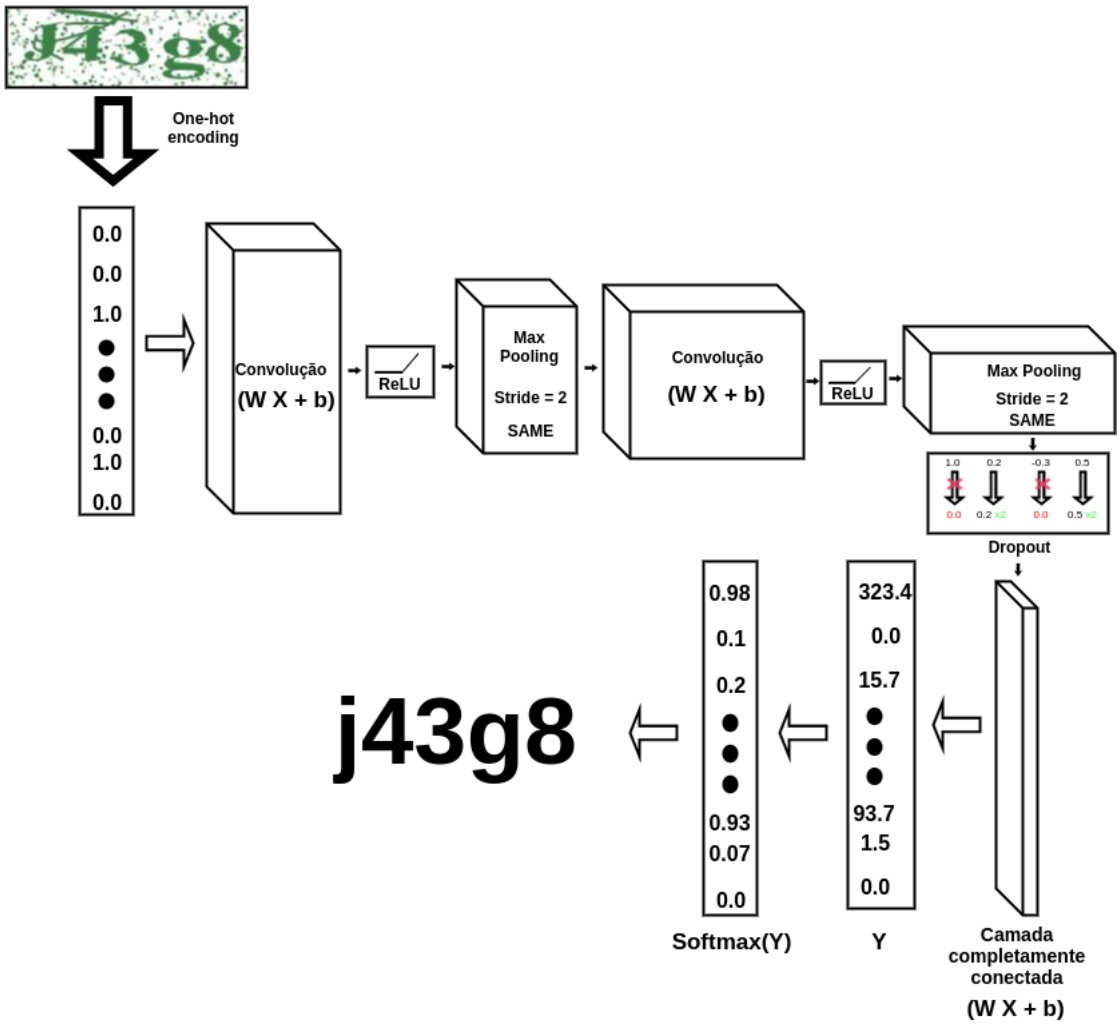

Figura 1. Exemplo da composição de todos os componentes presentes em redes neurais convolucionais de profundidade.

Experimento

Geração do Conjunto de dados

O conjunto de dados (ou *"dataset"*) que alimenta a rede neural é gerado em tempo de execução do treinamento. Cada imagem é lida de seu diretório em disco e carregada na memória como uma matriz de valores de pixel. Ao final deste processo há um vetor em memória com todas imagens existentes já pré-processadas. Isso é feito para o *dataset* de treinamento e de teste. Para o treinamento também será necessário um conjunto separado para teste que não possui nenhuma imagem presente no conjunto de treinamento. O *dataset* de treinamento terá cerca de 96% das imagens, e o *dataset* de testes terá 4% das imagens.

Junto com a geração do conjunto de dados, ocorre o pré-processamento das imagens. Durante o pré-processamento as imagens são transformadas para uma representação em escala de cinza. Por fim as imagens são redimensionadas para um tamanho menor, tornando os cálculos mais eficientes durante o treinamento da rede neural.

Treinamento

Após gerado o conjunto de dados, é possível trabalhar no treinamento do modelo da rede neural. Para isso será usado o *framework* ***TensorFlow***[TensorFlow] destinado à *Deep Learning*. Também será desenvolvido um *script* em *Python* que fará uso das funções disponibilizadas pela biblioteca do *TensorFlow*. Assim realizando o treinamento até atingir um valor aceitável de acerto no conjunto de teste. O resultado do treinamento será um arquivo binário representando o modelo que será utilizado para avaliação posteriormente.

Avaliação de acurácia

Com uma nova amostra de imagens, será feita a execução do teste do modelo que obteve melhor performance durante o treinamento. Ao final da execução será contabilizado o número de acertos e comparado com o número total da amostra de imagens para avaliação. Resultando assim em uma porcentagem que representa a acurácia do modelo gerado.

Desenvolvimento

Para a construção e treinamento da rede neural foi implementado um script em *Python* que possui toda a arquitetura da rede descrita de forma procedural. O *framework TensorFlow* chama a arquitetura dos modelos de *Graph* (ou grafo, em português) e o treinamento da rede neural é feito em uma *Session*.

O projeto é composto por 5 tarefas de implementação:

- Desenvolvimento do leitor e processador do conjunto de dados.
- Desenvolvimento da função que monta a rede neural.
- Configuração da rede neural para otimização dos resultados.
- Desenvolvimento da etapa de treinamento da rede neural.
- Desenvolvimento da etapa de teste e acurácia do modelo da rede neural.

Testes

Treinamento com 200 mil iterações

Inicialmente é realizado um treinamento com 200 mil iterações. A fase de treinamento completa levou **1 hora 23 minutos e 54 segundos** para completar. Deve-se salientar que para o primeiro treinamento há uma espera maior devido ao *caching* dos dados. Isso é feito pelo sistema operacional para otimizar a memória da GPU e do sistema em geral quando os dados são carregados para a memória volátil.

Como é possível observar nos gráficos o valor da acurácia fica em torno de 5% até um momento que começa a subir. Ao final do treinamento foi alcançado um valor máximo de acurácia igual a **84,38%** no conjunto de treinamento, **79,6%** no conjunto de teste.

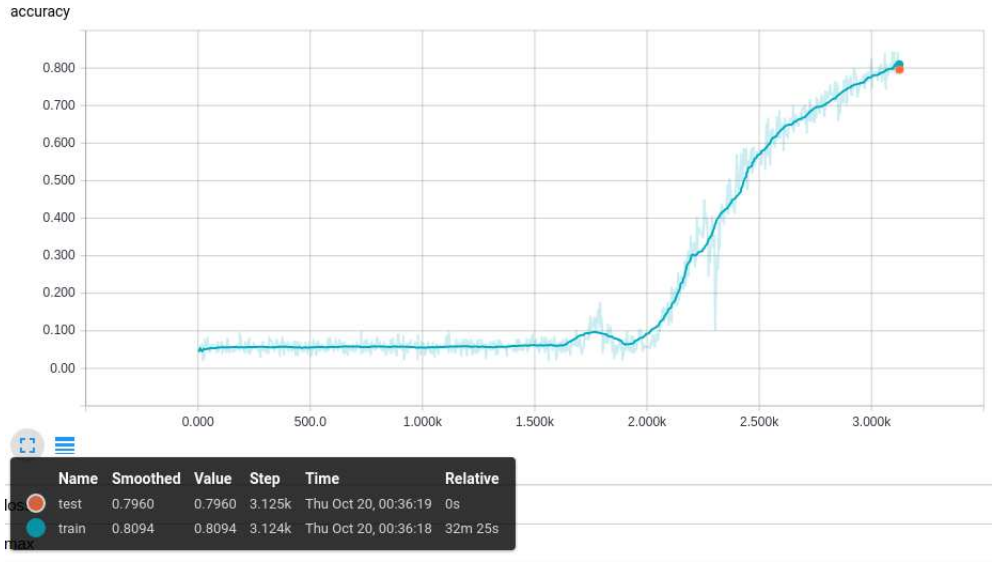

Figura 2. Gráfico da acurácia em relação ao número de passos para o treinamento da rede com 200 mil iterações.

Treinamento com 500 mil iterações

Visto a instabilidade nos valores de gráficos no treinamento anterior, a tentativa seguinte foi aumentar o número de iterações para 500 mil. O tempo total de treinamento foi de **1 hora 18 minutos e 23 segundos**.

Ao final do treinamento foi alcançado o valor máximo de acurácia igual a **98,75%** no conjunto de treinamento, **81,37%** no conjunto de teste.

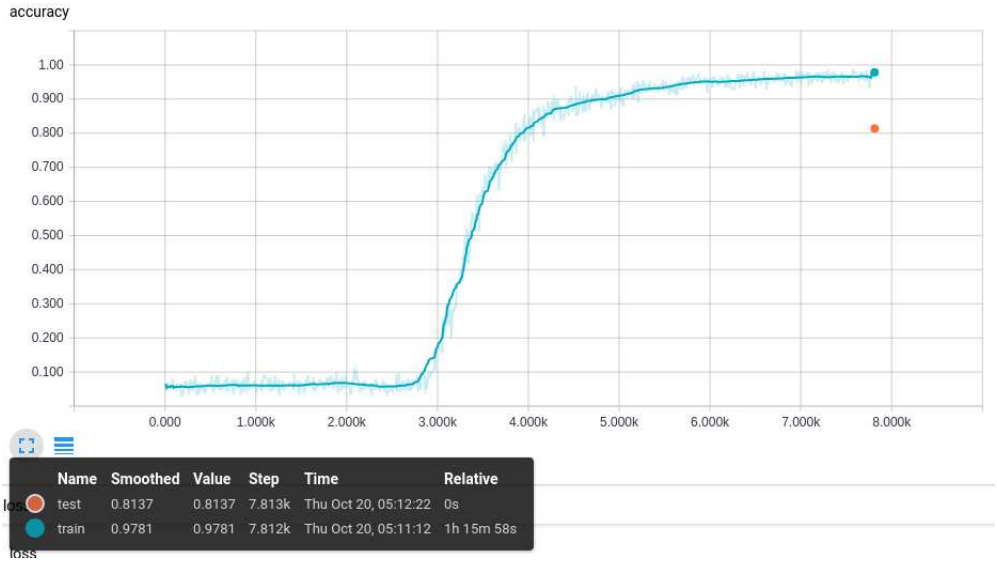

Figura 3. Gráfico da acurácia em relação ao número de passos para o treinamento da rede com 500 mil iterações.

Treinamento com 500 mil iterações e Dropout de 50%

Na tentativa de minimizar os problemas encontrados anteriormente, foi realizado um terceiro treinamento. Foi visto que uma das técnicas de regularização para minimizar o

overfitting é adicionando uma camada de *dropout* ao modelo. Nossa arquitetura já previa uma camada de *dropout*, no entanto o parâmetro de probabilidade de mantimento das ativações estava configurado para 75% (0,75). Para o terceiro treinamento foi configurada a probabilidade do *dropout* para 50% (0,5) e assim analisados os resultados. O tempo total de treinamento foi de **1 hora 18 minutos e 53 segundos**.

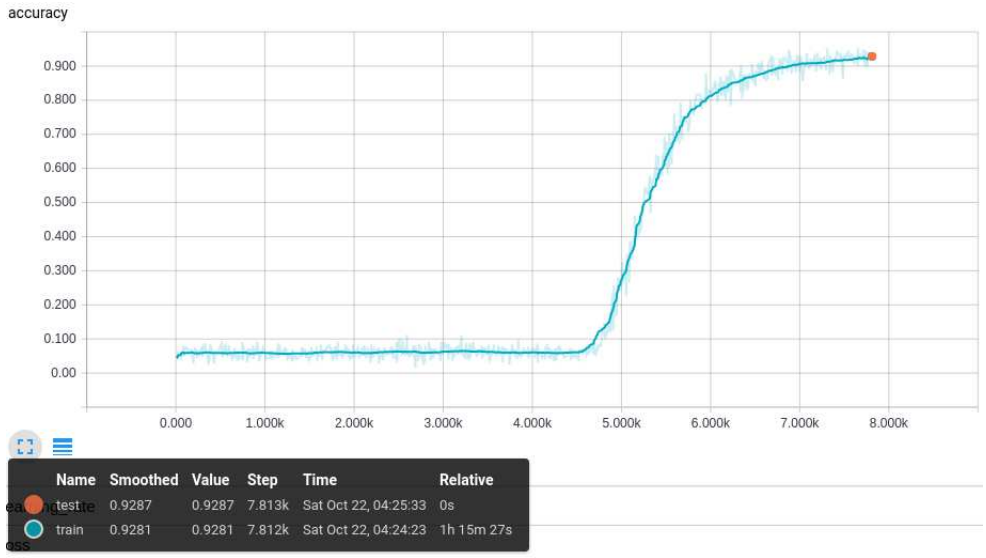

Figura 4. Gráfico da acurácia em relação ao número de passos para o treinamento da rede com 500 mil iterações e probabilidade de *dropout* igual a 50%.

Ao final do treinamento foi alcançado o valor máximo de acurácia igual a **95,31%** no conjunto de treinamento, **92,87%** no conjunto de teste.

Conclusão

Os testes realizados em todos os casos mostraram ser possível atingir um resultado razoável na tarefa de reconhecimento de textos em imagens, isso com poucos ajustes à configuração de treinamento de redes neurais. Atualmente a quantidade de exemplos e tutoriais disponíveis para tarefas de aprendizado de máquina é imenso. Fica claro que é possível implementar classificadores mesmo com poucos recursos.

Diante do objetivo alcançado pelo trabalho, fica aparente que fontes públicas de dados podem estar vulneráveis.

Trabalhos futuros

Como possíveis trabalhos futuros, cita-se:

- Fazer um melhor uso das informações geradas pelo processo de treinamento para gerar heurísticas mais inteligentes. Um exemplo seria utilizar outros tipos de otimizadores para a função da perda.
- Estender o sistema para realizar o reconhecimento de outros tipos de CAPTCHAs.
- Estender o sistema para realizar o reconhecimento de tipos de CAPTCHAs que possuem um tamanho de texto variável.
- Realizar um estudo sobre *Web crawlers* em fontes públicas que utilizam CAPTCHA, executando o sistema proposto neste trabalho.

- Implementar um sistema de reconhecimento de CAPTCHAs mais avançados que solicitam a classificação de uma cena completa ou identificação de objetos em imagens.
- Estudar um artifício mais efetivo para o bloqueio de consultas automatizadas em *websites*.

Considera-se de extrema importância a implementação de projetos desse tipo pois o mesmo auxilia na compreensão e aplicação de Inteligencia Artificial em casos específicos.

Referências

Bengio, I. G. Y. and Courville, A. (2016). Deep learning. Book in preparation for MIT Press.

Dumoulin, V., Visin, F., and Box, G. E. P. (2016). A guide to convolution arithmetic for deep learning.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. Acessado: 21/10/2016.

TensorFlow. TensorFlow — an Open Source Software Library for Machine Intelligence. Acessado: 03/08/2016.