Machine Learning Engineer Nanodegree

Capstone Project

Rafael Augusto Cupello Lopes
20 de Agosto de 2017

I. Definição

Overview do Projeto

Uma tática muito utilizada para proteção contra o acesso de bots a áreas mais críticas de um site é a utilização de CAPTCHAs, que consiste em um teste de desafio cognitivo onde o usuário é solicitado a digitar as letras e números presentes no conteúdo de uma imagem que contém algum tipo de "interferência" (como letras tortas, riscos, ...). Apesar do termo CAPTCHA ter sido utilizado pela primeira vez em 2003, essa técnica anti-spam já havia sido utilizada em 1997 pelos funcionários da AltaVista para prevenir bots de adicionarem URLs ao seu motor de busca web. Exemplos de CAPTCHAs:

Um problema que o CAPTCHA traz para o usuário bem intencionado é a dificuldade na automatização de certas tarefas como, por exemplo, a compra recorrente em um site com CAPTCHA que poderia ser acionada por um alarme mensal ou algo do gênero. Buscando facilitar a utilização de bots benignos e preservar a cultura da automatização, construirei ao longo deste trabalho uma solução base para apenas um dígito mas facilmente extensível e aplicável à esse cenário.

Definição do Problema

Durante o projeto foi criado um modelo capaz de solucionar, com uma margem de erro de no máximo 1%, o número no centro de uma imagem com CAPTCHA gerada a partir de um padrão pré-definido; o padrão consiste em uma imagem com ruídos (linhas, pontos...) e contém de 2 a 3 números sendo o número do centro o label a ser identificado pelo modelo.

Tarefas a serem executadas durante esse projeto:

1. Utilizar a biblioteca python captcha para gerar o dataset de treino e teste.;
2. Pré-processar tanto o dataset de treino quando o de teste, normalizando seus dados;
3. Salvar os dados pré-processados em batchs para facilitar o treino, otimizando o gasto de memória;
4. Treinar o classificador capaz de determinar o número presente no centro da imagem;
5. Salvar o classificador para usos posteriores.

Métricas

A métrica escolhida para medir o quão bem o modelo está performando é a Acurácia, que consiste na proporção de itens classificados corretamente (para esse problema respostas parcialmente corretas não são consideradas soluções válidas). A fórmula de Acurácia consiste em:

$$Acurácia = \left(\frac{Número\ de\ Itens\ Classificados\ Corretamente}{Número\ total\ de\ itens}\right)$$

Essa métrica é muito utilizada em casos onde o número de classes está balanceado, como nesse projeto. Em cenários com classes desbalanceadas, como por exemplo de eventos raros, essa métrica é nociva ao modelo. Exemplo: em um cenário onde o cliente tem apenas 10% de probabilidade de comprar; para atingir uma boa performance medida por acurácia, prevendo se ele vai comprar ou não, bastaria a previsão de que o cliente nunca compra para atingir 90% de acerto, levando o modelo a não aprender ou aprender erroneamente.

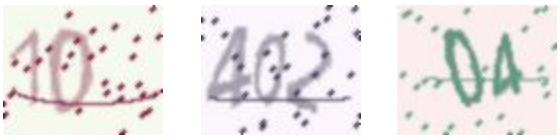
II. Análise

Explorando os Dados

Para esse projeto foram gerados dois datasets de imagens (teste e treino) utilizando-se da biblioteca python captcha. Cada uma das imagens tem como dimensão 65x80, são coloridas no formato RGB (ou seja, possuem 3 camadas de cores) e todas possuem algum tipo de ruído que respeita às seguintes regras: os número podem estar inclinados mas nunca ao ponto de atingir 180 graus; ruídos como pontos e linhas podem estar presentes nas imagens mas desde que não cubram o número por completo.

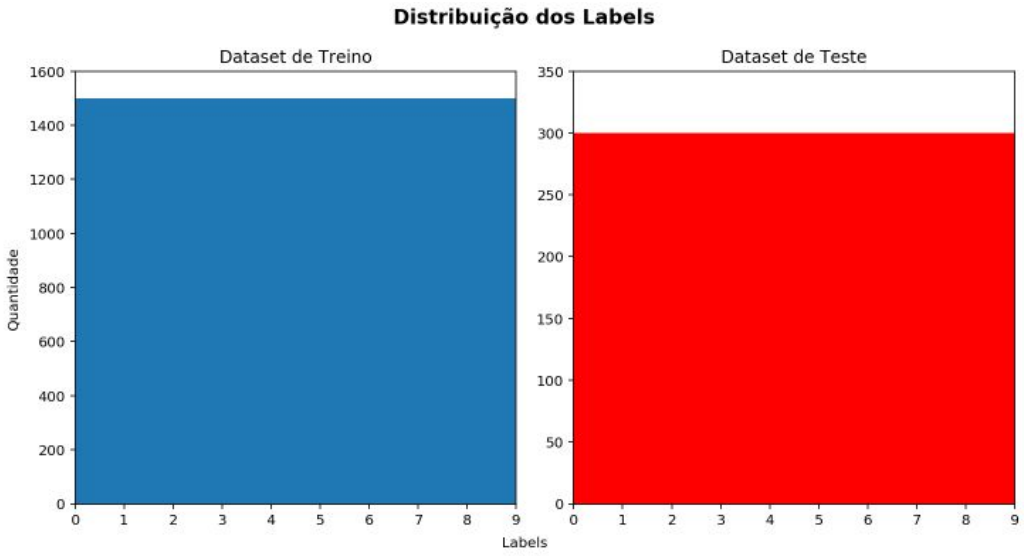
O dataset de treino consiste em: 5.000 imagens com o label cercado por dois dígitos, 5.000 imagens com o label com um dígito à sua direita e 5.000 imagens com o label com o dígito à sua esquerda, sendo o label fazendo parte da sequência [0..9], totalizando 15.000 imagens de treino. O dataset de teste é composto da mesma maneira, alterando-se apenas a quantidade de 5.000 para 1.000, totalizando 3.000 imagens de teste. O label do modelo está salvo na primeira parte do nome da imagem; exemplo: A imagem 0_2_402.png tem como o valor 0 (zero) o seu label.

Exemplo de imagens geradas (com o número zero como label):



Análise Exploratória Visual

Os gráficos abaixo comprovam a distribuição de dados citada anteriormente:



Algoritmos e Ténicas

Para o problema em questão, processamento e extração de informação de imagens, uma das técnicas mais utilizadas e prestigiadas pelo seu sucesso são as CNNs (convolutional neural network). As CNNs são redes neurais que, compartilhando a informação aprendida a cada convolução conseguem, por exemplo, identificar objetos iguais mesmo que estejam localizados em pontos diferentes. A CNN utilizada neste projeto é composta pelas seguintes camadas: Convolutional Layer, Pooling Layer, Flatten Layer ReLu Layer, Fully Connected Layer e Output Layer.

- **Convolutional Laye**r: Principal camada da CNN, é onde acontece um número específico de operações de convolução. A Operação de Convolução consiste em transformar a imagem em outra imagem através do uso de filtros, que analisam patches da imagem utilizando sempre os mesmos pesos.
- **Pooling Layer**: Layer responsável por "compactar" (downsampling) uma imagem, reduzindo assim suas dimensões, objetivando diminuir o tempo de processamento. Existem diversos algoritmos de pooling, sendo o algoritmo chamado "max pooling" o escolhido e utilizado neste projeto.
- **Flatten Layer**: Layer responsável por "achatar" (flatten) as dimensões da imagem.
- **Relu Layer**: Layer onde é aplicado a função de ativação f(x)= max(0,x).
- **Fully Connected Layer / Output Layer:** Layer não-convolucional onde todos os neurônios de entrada se conectam com todos os neurônios de saída. A última *Fully Connected Layer* também é conhecida como *Output Layer*; é na *Output Layer* que ocorre o mapeamento dos dados de entrada para os labels de saída.

Benchmark

O benchmark escolhido para esse projeto é atingir 99% de Acurácia nos dados de teste. O nível de assertiva é alto pois objetivamos criar um modelo que possa ser absorvido por modelos maiores que

busquem classificar corretamente todos os dígitos presentes em um CAPTCHA, (o que seria inviável caso o modelo base esteja com uma acurácia mediana).

III. Metodologia

Pré-Processamento de Dados

O pré-processamento de dados de treino consiste nos seguintes passos:

1. A lista de imagens é embaralhada;
2. A lista de imagens é dividida em 9 batchs;
3. Cada batch é dividido da seguinte forma: 90% dos dados destinados ao treino e 10% à validação;
4. Os 9 batchs e o batch de validação tem as features de cada uma de suas imagens normalizadas;
5. Cada batch é salvo no formato pickel para facilitar o uso posterior.

O pré-processamento de dados de teste consiste nos seguintes passos:

1. A lista de imagens é embaralhada;
2. As features de cada uma de suas imagens normalizadas;
3. O resultado é salvo no formato pickel para facilitar o uso posterior.

Implementação do Modelo

Sem qualquer complicação e implementada com auxílio da ferramenta open-source TensorFlow, desenvolvida pela Google, a arquitetura final da CNN se apresenta da seguinte forma:

$$INPUT \Rightarrow [CONV \Rightarrow POOL]^{3} \Rightarrow FLAT \Rightarrow [DROP \Rightarrow FC]^{3} \Rightarrow DROP \Rightarrow OUTPUT$$

A tabela a seguir mostra detalhadamente cada uma das camadas:

Camada	Descrição
Input Layer	15.000 imagens com dimensões 65x80x3
Convolutional Layer 1	Filtros: 32; Dimensão do filtro: 3x3; Stride: 1x1; Padding: SAME
Pooling Layer 1	Dimensão: 2x2; Stride: 2x2; Padding: SAME
Convolutional Layer 2	Filtros: 64; Dimensão do filtro: 3x3; Stride: 1x1; Padding: SAME
Pooling Layer 2	Dimensão: 2x2; Stride: 2x2; Padding: SAME
Convolutional Layer 3	Filtros: 128; Dimensão do filtro: 3x3; Stride: 1x1;

	Padding: SAME
Pooling Layer 3	Dimensão: 2x2; Stride: 2x2; Padding: SAME
Flatten Layer	-----
Dropout Layer 1	Probabilidade: 60%
Fully-Connected Layer (com ReLu) 1	Número de Outputs: 512
Dropout Layer 2	Probabilidade: 60%
Fully-Connected Layer (com ReLu) 2	Número de Outputs: 256
Dropout Layer 3	Probabilidade: 60%
Fully-Connected Layer (com ReLu) 3	Número de Outputs: 128
Dropout Layer 4	Probabilidade: 60%
Output Layer	Número de Outputs: 10

Refinamento

Apesar da CNN possuir diversos parâmetros ajustáveis, o resultado desejado foi alcançado em pouco tempo. A única alteração feita foi aumentar o número de epochs de 10 para 20, o que resultou em um ganho de 3% de performance na acurácia no dataset de validação (os logs dos resultados encontram-se explicitados no arquivo projeto_final.ipynb).

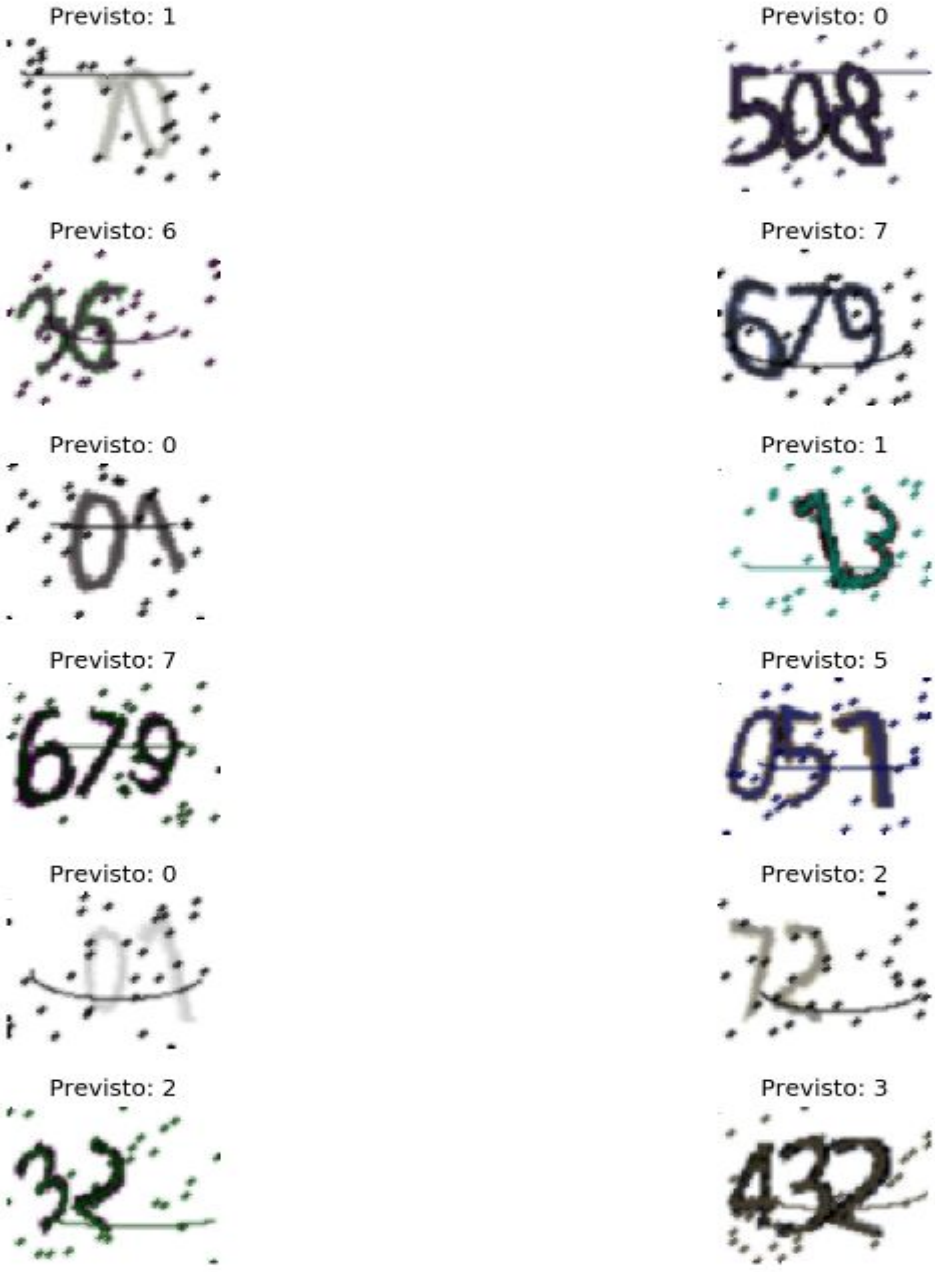
IV. Resultados

Avaliação do Modelo e Validação

Durante toda a fase de treino um dataset auxiliar de validação foi utilizado para verificar a performance da CNN à medida que um novo epoch era concluído. A acurácia na validação iniciou-se em 13% (epoch 1) e concluiu atingindo 99 (epoch 20). No dataset de teste a acurácia atingida foi de 99%.

O algoritmo otimizador utilizado foi o Adam Optimization Algorithm; conhecido no campo do Deep Learning por conseguir ótimos resultados rapidamente, Adam O. Algorithm é um bom substituto ao muito conhecido SGD (Stochastic Gradient Descent) por combinar o melhor entre os algoritmos AdaGrad e RMSProp, provendo um algoritmo otimizador capaz de lidar com problemas que envolvam ruídos.

Segue abaixo algumas das classificações feitas corretamente pelo modelo para o número no centro de uma imagem com CAPTCHA:



Classificações Corretas

IV. Conclusão

Sumário

Os passos executados ao longo desse projeto que me levou ao resultado desejado foram:

1. Levantamento de um problema relevante;
2. Criação dos datasets tanto de treino quanto de teste, com CAPTCHAs no formato pré-definido;
3. Embaralhamento e separação do dataset de treino em batchs;
4. Pré-processamento das imagens, onde normaliza-se cada uma de suas features;
5. Criação da CNN;
6. Treinamento do Modelo;
7. Validação do Modelo.

Reflexão e Melhorias

Com o auxílio de uma plataforma dedicada ao Deep Learning para executar o projeto, no caso o FloydHub, consegui atingir a meta esperada em pouco tempo de processamento, o que foi muito reconfortante. Porém desvendar um CAPTCHA real envolve acertar todos os dígitos e caracteres presentes na imagem. Por isso, para o futuro, vejo duas principais melhorias/adições: criação de um modelo também capaz de identificar letras; criação de um modelo capaz de "recortar" sub-imagens para cada um dos símbolos presentes na imagem original, com os mesmos no centro, para que então possa-se criar um modelo composto com o criado ao longo desse projeto capaz de resolver todos os símbolos presentes em um CAPTCHA.

Referências

- https://en.wikipedia.org/wiki/CAPTCHA
- https://pypi.python.org/pypi/captcha
- http://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
- https://www.floydhub.com
- https://www.tensorflow.org/tutorials/layers
- https://en.wikipedia.org/wiki/Convolutional_neural_network