APLICAÇÃO DE APRENDIZAGEM SUPERVISIONADA NO TREINAMENTO DE SISTEMAS ROBÓTICOS

Rafael de Mattos (PROVIC), Ariangelo Hauer Dias (Orientador), e-mail: ariangelo@gmail.com.

Universidade Estadual de Ponta Grossa/ Departamento de Informática.

Engenharias, Engenharia de Computação

Palavras-chave: Aprendizado de máquina, reconhecimento de padrões, inteligência artificial.

Resumo

Neste projeto foi realizado o programa para o controle de um robô capaz reconhecer letras e formar palavras. O software irá ajudar o robô a formar palavras por meio do aprendizado de máquina (*Machine Learning*) reconhecendo os padrões de cada letra. O robô fará parte de um projeto feito por outro orientando do mesmo orientador.

Introdução

Robôs são dispositivos autônomos ou remotamente controlados capazes de interagir com o meio. Dependendo da aplicação, eles podem ser compostos por inúmeras peças, sensores e módulos. Utilizando o aprendizado de máquina é possível dar certa inteligência à máquina, isso faz com que gere mais autonomia por parte do robô. Esse tipo de sistema que será projetado nessa pesquisa pode ser aplicado em vários cenários industriais onde se utiliza computadores, braços robóticos e demais maquinários capazes de fazer movimentos no espaço.

No âmbito atual da computação a análise dos dados possui extrema importância, conquistando um grande poder em muitas aplicações. Neste projeto será realizado uma aplicação que une a computação com a análise de dados e reconhecimento de padrões em imagens, tornando um robô capaz de formar palavras como uma pessoa.

Usando como ferramenta de aprendizado de máquina e reconhecimento de padrões a técnica de redes siamesas que faz parte do tópico de redes neurais em aprendizado de máquina.

Material e Métodos

No desenvolvimento desse projeto foi utilizado a linguagem de programação Python na versão 3.6.8 pelo motivo de possuir maior número de ferramentas existentes para o objetivo desse projeto. Além disso foi utilizado uma rede neural pré-treinada desenvolvida pela *Google Inc.* chamada de Tesseract OCR. Para conseguir a localização de cada caractere situado na imagem processada foi necessário a utilização da ferramenta OpenCV3 disponível para a linguagem Python.

Após o levantamento bibliográfico sobre o assunto foi dado início ao processo de detecção de caracteres. O algoritmo da rede pré-treinada capaz de detectar cada letra executando os seguintes passos:

1. Quando o arquivo é lido procura-se na imagem os contornos de cada letra nos quais são guardados. O algoritmo possui maior facilidade em reconhecer caracteres quando a imagem é branco no preto, portanto todos os contornos presentes na imagem são juntados e salvos em uma estrutura de dados chamada Blobs;
2. Os blobs armazenam os contornos como se fossem várias linhas de texto organizadas por espaçamento, posteriormente eles serão repartidos em palavras de acordo com o espaçamento específico de cada letra e por final essas as letras de cada palavra são separadas individualmente;
3. Após os dados serem guardados começa o processo de reconhecimento que é dividido em duas etapas;
4. A primeira etapa se concentra em coletar as palavras para um classificador, essas palavras são usadas como um treinamento para esse classificador que quando atinge uma quantidade considerável de imagens passadas pelo treinamento a segunda etapa é iniciada;
5. A segunda parte é coletado apenas imagens que foram mal reconhecidas ou não identificadas, estas são passadas novamente para classificador, para que ele desta vez possa reconhecê-las e refinar ainda mais o método de classificação. Com isso, quando o classificador é treinado e obtém bons pontos de acerto ele pode passar para fase de teste onde novas imagens são passadas pelo classificador e por fim entra em fase de produção;
6. O algoritmo retorna dados do tipo `char` com as letras de cada palavra separadas.

Para melhor entendimento de como funciona o método de classificação utilizado pelo Tesseract OCR funciona, a seguir será explanado como funciona a rede siamesa usada.

Uma rede siamesa é um tipo específico de arquitetura de rede neural, ela “aprende” através da diferença entre duas entradas de informações, mais especificamente ela faz uma comparação entre as duas entradas e relaciona o que há de mais parecido entra elas. Esse modelo de arquitetura consiste em duas redes neurais idênticas onde cada uma coleta uma imagem (entrada de informação) onde será coletado as características da imagem como cor, brilho, contraste, saturação, ângulos e tamanho. Na última camada de cada rede encontra-se uma função que realiza a comparação entre as características das imagens.

Essa função que realiza a comparação das características funciona da seguinte forma:

1. Considerando que a saída de cada rede neural é um vetor onde cada posição desse vetor corresponde ao valor extraído de cada característica da imagem a função de comparação irá trabalhar em cima desses dois vetores;
2. Quando coletadas os vetores ocorre uma comparação entre cada

ISSN 1676-0093

elemento do vetor que irá gerar uma saída que iremos chamar de *Y* que é um vetor cujo irá conter somente valores binários onde 0 corresponde a muita semelhança entre os elementos do vetor e 1 para quando os elementos forem significativamente distintos.

3. Depois da etapa 2 é feito a distância euclidiana entre os dois vetores de saída. A equação da distância euclidiana é mostrada em (2), onde *X1* e *X2* são as entradas (característica das imagens) e *Gw* é a função que define a saída dessas características que corresponde ao vetor de característica da imagem.
4. Com esses três vetores e o valor da distância euclidiana será utilizada a função de comparação representada em (3). A variável *Dw* equivale a distância euclidiana calculada anteriormente, *Y* é o resultado da comparação entre as características de *X1* e *X2*. A função *max* é usada para encontrar o maior valor entre 0 e *(m – Dw)* onde *m* é o valor da margem na qual deve ter um valor maior que 0. Quando temos um valor de margem maior do que zero significa que há um valor limite no qual indicará até onde os valores rotulados como *distintos* irão entrar no cálculo. Na Figura 4, podemos ver o gráfico de exemplo onde termos uma curva azul que representa a função de comparação usando os pares que são distintos e a linha vermelha que representa a função usando os pares que não são distintos significativamente.

$$\sqrt{\{G_W(X_1) - G_W(X_2)\}^2} \tag{2}$$

$$(1-Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{max(0, m - D_W)\}^2 \tag{3}$$

Após todo o processo de reconhecimento de caracteres foi elaborado uma função em que gera palavras a partir das letras já existentes na imagem.

Com isso para obter palavras derivadas das letras que foram reconhecidas foi necessário coletar o vetor de *strings* e dividi-lo por palavras e remover os caracteres especiais como “! @ # $ % ¨ & *” e depois utilizar a biblioteca para a linguagem Python chamada NLKT que possui um dicionário de palavras de vários idiomas. As palavras são formadas aleatoriamente com letras posicionadas aleatoriamente que ao passarem por uma expressão condicional são salvas como válidas ou não.

Resultados e Discussão

Após a fase de criação do programa foi feito uma análise sobre como melhorar o reconhecimento dos caracteres em uma imagem. Por mais que o algoritmo de reconhecimento fosse pré-treinado ainda existia falhas em alguns tipos de entrada (imagens) que não conseguiam ser lidos pela rede neural.

Várias etapas podem ser feitas para melhorar a fidelidade da ferramenta**,** tais como: Redimensionamento, binarização da imagem e rotação/alinhamento das linhas.

- **Redimensionamento**: A taxa de erro de OCR foi mais fortemente correlacionada à altura de uma letra maiúscula em pixels, independentemente do DPI ou do tamanho do ponto. Veja o enredo abaixo. Os erros mais comuns quando a altura da letra ficou muito grande eram dependentes de uma fonte específica, mas incluíam interpretar i como 1 (mais comum com fontes serifadas), confundir um ponto-e-vírgula com dois-pontos e interpretar uma letra minúscula como é letra maiúscula (por exemplo, i / I, s / S, o / o, k / k).
- **Binarização da imagem:** o processo de binarização de uma imagem se resume a deixar as cores preto e branco da imagem. Isso faz com que a detecção das bordas das letras e seus limites de dimensão sejam melhor definidos. Em certos casos em que haja sombra na imagem esse método não será a melhor escolha pois a cor preta pode ficar por cima das letras que estejam na sombra. Para isso é desejado que tenha uma boa iluminação na imagem para binariza-la.
- **Rotação/Alinhamento das linhas:** Uma imagem inclinada é quando uma página foi digitalizada quando não está reta. A qualidade da segmentação de linha da Tesseract reduz significativamente se uma página estiver muito distorcida, o que afeta severamente a qualidade do OCR.

Conclusão

Com esse projeto desenvolvido acreditasse que possa ser utilizado no próximo projeto do outro orientando para a implementação em um robô inteligente que possa formar palavras a partir de letras espalhadas no chão ou na base em que o robô esteja. A taxa de acerto utilizando os métodos de tratamento de imagem estudados possuem bons resultados que facilitam o reconhecimento no momento em que a imagem seja retirada do ambiente.

Agradecimentos

Gostaria de agradecer o meu professor orientador Ariangelo Hauer Dias que pode fazer a alteração do tema do projeto que foi estudado. Também agradeço o orientado Ezequiel Orloski que me ajudou a relacionar o meu trabalho com o trabalho do robô que ele está desenvolvendo.

Referências

GÉRON, Aurélien. **Hands-on Machine Learning with Scikit-Learn & Tensorflow**, 2017, First Edition.

Repositório Tesseract OCR - https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality

HADSELL, R., CHOPRA, S., LECRUN, Y - **Dimensionality Reduction by Learning an Invariant Mapping**, 2006.

SMITH, R - **An Overview of the Tesseract OCR Engine**

GUPTA, H. **One Shot Learning with Siamese Networks in PyTorch** - hackernoon.com/one-shot-learning-with-siamese-networks-in-pytorch-8ddaab10340e