XVI CEEL - ISSN 2178-8308
Universidade Federal de Uberlândia
05 a 09 de novembro de 2018

DESENVOLVIMENTO DE UM SOFTWARE PARA RECONHECIMENTO E LEITURA DE TEXTO UTILIZANDO REDES NEURAIS ARTIFICIAIS

Leonardo Muttoni*[1]

[1]FEELT - Universidade Federal de Uberlândia

***Resumo* - Este trabalho apresenta o desenvolvimento de um software para o reconhecimento de textos a partir de fotografias de páginas de texto impresso. O software utiliza uma rede neural artificial para o reconhecimento de caracteres e faz a síntese de fala do texto reconhecido. O funcionamento de cada etapa de processamento é exposto e os resultados preliminares obtidos até então são apresentados.**

***Palavras-Chave*- Reconhecimento óptico de caracteres, Redes Neurais, Síntese de fala, Tecnologia Assistiva.**

DEVELOPMENT OF A SOFTWARE FOR RECOGNITION AND SPEECH OF TEXT USING ARTIFICIAL NEURAL NETWORKS

***Abstract* - This paper presents the development of a software for the recognition of texts contained in a photograph of a page of printed text. The software uses an artificial neural network for character recognition and makes speech synthesis of recognized text. The operation of each processing step is exposed, and some preliminary results obtained so far are presented.**

***Keywords* - Neural Networks, Optical Character Recognition (OCR), Speech synthesis, Assistive technology.**

I. INTRODUÇÃO

O reconhecimento automático de texto tem se tornado cada vez mais usual em diversas áreas como na educação (bibliotecas e repositórios institucionais), na área bancária/empresarial (digitalização de cheques, faturas ou outros documentos financeiros/contábeis/tributários), em sistemas de saúde (digitalização de registros médicos), judiciais (digitalização de processos) e na segurança pública (reconhecimento de placas de veículos) [1].

Apesar se ser um campo já bem explorado, ainda há desafios no reconhecimento de textos a partir de imagens de baixa qualidade, como quando adquiridas sob iluminação inadequada e com distorções geométricas – como por exemplo a ondulação de uma folha segurada à mão. Tais cenários são comuns ao se considerar o uso de um *smartphone* na aquisição livre destas imagens por usuários leigos.

*muttoni@gmail.com

O presente trabalho apresenta o desenvolvimento de um software para reconhecimento de texto e sua posterior pronunciação através de um sintetizador de voz. A entrada do sistema é uma fotografia de uma página de texto impresso que é submetida às diversas etapas de processamento de imagem. O reconhecimento dos caracteres isolados do texto é feito através de redes neurais artificiais e a pronunciação do texto reconhecido é realizado por um sintetizador de fala de terceiros, integrado ao sistema desenvolvido.

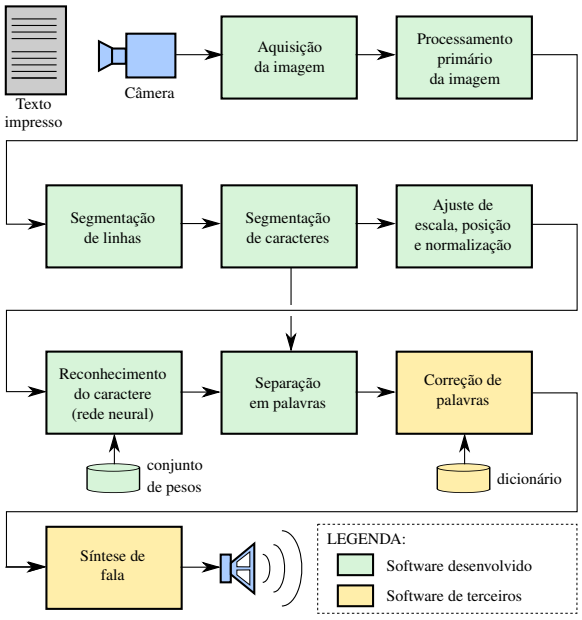
Este software pode ser utilizado para leitura automática de texto, auxiliando por exemplo pessoas com necessidades especiais visuais, se caracterizando como uma tecnologia assistiva [2].

As subseções da Seção II detalham o funcionamento de cada etapa de processamento do software. Já a Seção III apresenta alguns resultados preliminares deste desenvolvimento com suas discussões.

II. DESENVOLVIMENTO

A Figura 1 apresenta o diagrama de blocos do software desenvolvido.

Figura 1: Etapas de processamento



As subseções seguintes detalham o fluxo de processamento de cada um dos blocos, da imagem do texto até a síntese da fala.

A. Aquisição da imagem

A imagem do texto a ser reconhecido pode dar entrada no programa de duas formas: (i) apontamento manual do arquivo de imagem; (ii) fotografia através de um *smartphone* e envio automático para o computador para o reconhecimento.

Os formatos de imagens suportados são: JPEG, PNG, TIFF, BMP e GIF. Os testes feitos neste trabalho se basearam em fotografias de câmera de *smartphone* com resolução 3264×2448. É recomendado que a fonte do texto fotografado tenha no mínimo 16 *pixels* de altura.

B. Processamento primário da imagem

O processamento primário envolve as seguintes operações:

1) Conversão da imagem para tons de cinza

A imagem de entrada normalmente é uma fotografia colorida, contendo os canais RGB (*Red, Green, Blue*). Esta etapa do processamento converte os canais RGB para o canal único de luminância I, transformando a imagem para tons de cinza, conforme a Equação 1.

$$I = 0{,}298936R + 0{,}587043G + 0{,}114021B \tag{1}$$

2) Esticamento do contraste da imagem

A imagem na entrada desta etapa do processamento normalmente possui valores comprimidos em uma faixa que depende das condições das configurações da câmera e da iluminação do ambiente no momento da captura da fotografia. O objetivo desta etapa é esticar o contraste da imagem, tornando os pixels escuros ainda mais escuros e os claros na direção oposta. Para minimizar a influência dos ruídos neste processo, foi adotado um descarte de 1% dos pixels mais claros e de 1% do pixels mais escuros da imagem antes do cálculo do limite inferior (a) e superior (b) de esticamento. A operação se dá conforme a Equação 2, que converte a imagem I em G.

$$G = \frac{I - a}{b - a} \tag{2}$$

3) Detecção da polaridade da imagem

A imagem fotografada pode possuir tanto o texto mais escuro que o fundo, quando o oposto. Para lidar com estas duas situações possíveis, foi elaborada a detecção de polaridade da imagem através da verificação do valor médio de todos os pixels da imagem, m_I, com o valor normalizado de 0,5, conforme a Equação 3. Caso seja detectado que o texto é mais claro que o fundo, é feita a inversão da imagem.

$$I = \begin{cases} 1 - I, & \text{se } m_I < 0{,}5 \\ I, & \text{caso contrário} \end{cases} \tag{3}$$

4) Uso do limiar adaptativo

A imagem em tons de cinza é submetida ao algoritmo de Bradley [3], que faz a *binarização* da imagem através da comparação do valor de cada pixel com a média de seus pixels vizinhos em uma janela 128×128. Esta etapa do processamento é muito importante pois separa o texto do fundo de forma confiável mesmo com diferenças de iluminação ou tonalidade em partes da imagem.

5) Remoção de ruído tipo sal e pimenta

A última etapa do processamento primário da imagem visa remover ruídos impulsivos que normalmente se tornam aparentes após o processo de limiarização. Foi adotado um filtro de mediana com vizinhança 3×3.

C. Segmentação de linhas

A imagem na entrada deste processo é uma imagem monocromática, binarizada, com o texto em branco (1) e o fundo em preto (0). Esta etapa do processamento procura detectar e isolar as linhas de texto presentes na imagem.

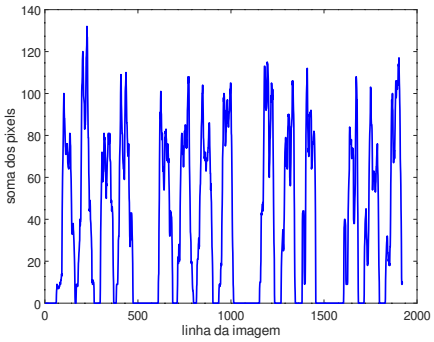
Para lidar com situações como uma leve inclinação do texto, amassamento da folha de texto ou a distorção do tipo barril, a segmentação de linha é executada dividindo a imagem em n *tiras* verticais.

Em cada tira T_k o vetor $\mathbf{s}$ é calculado a partir da soma ao longo das linhas da imagem daquela tira, conforme a Equação 4.

$$\mathbf{s} = G(x) = \sum_y T_k(x,y) \tag{4}$$

Um exemplo deste vetor é apresentado na Figura 2. Os trechos que contém linhas de texto naquela tira, chamados de *marcas*, são então inferidos através da limiarização de $\mathbf{s}$.

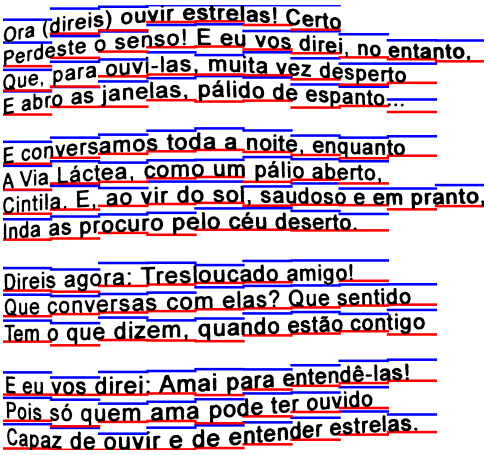
Figura 2: Exemplo de vetor $\mathbf{s}$



O algoritmo de limiarização utiliza como parâmetros, além do limiar, uma altura mínima para as marcas e também para os espaços. Isto evita com que ruídos na imagem provoquem falsas detecções de linhas ou espaços.

A Figura 3 apresenta um exemplo de segmentação das linhas e texto de uma imagem utilizando 10 tiras. É possível verificar que este método é capaz de encontrar as fronteiras corretas das linhas mesmo com distorções geométricas na imagem.

Figura 3: Exemplo de segmentação das linhas utilizando 10 tiras



D. Segmentação de caracteres

Cada linha segmentada é submetida ao segmentador de caracteres. Este faz a limiarização da soma dos valores dos pixels ao longo das colunas da imagem de cada linha de texto, para estabelecer a fronteira entre cada caractere. Este método escolhido é simples e necessita um espaçamento claro entre os caracteres.

A Figura 4 exibe as fronteiras detectadas entre os caracteres de uma linha do texto.

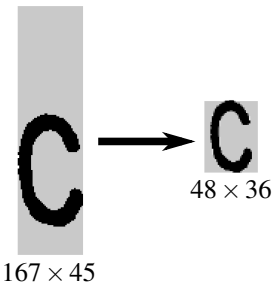
Figura 4: Exemplo de segmentação de caracteres

E. Ajuste de escala, posição e normalização

Os caracteres segmentados possuem um tamanho arbitrário, enquanto a próxima etapa de processamento espera um tamanho de entrada fixo, de 48×36. Portanto, é feito um ajuste na escala da imagem (redução ou ampliação) mantendo a razão de aspecto original.

O caractere é alinhado ao centro da imagem e as linhas ou colunas que sobrarem são preenchidas com o valor zero. Uma desvantagem deste método simplório é que alguns caracteres distintos, como “o” e “O”, após a operação de escala para preencher a saída de 48×36, acabam ficando com a mesma aparência, podendo confundir a etapa de reconhecimento. A Figura 5 apresenta um exemplo de operação desta etapa.

Figura 5: Operação de ajuste de escala e posição



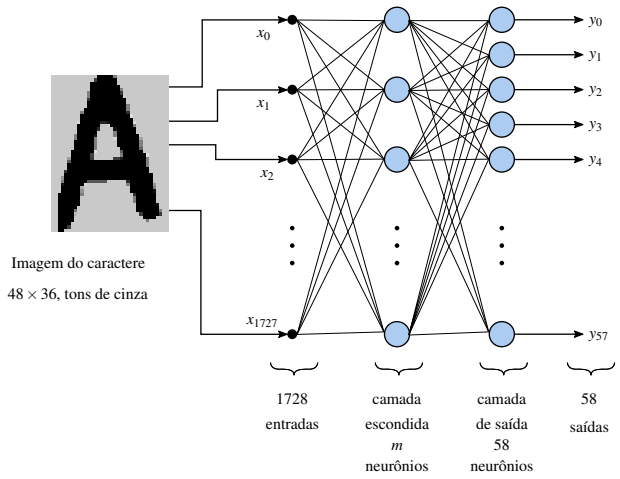
Após a operação de escala, devido às interpolações envolvidas é possível que alguns pixels fiquem com valores fora da faixa 0 — 1. Para restaurar esta faixa é feita uma nova normalização na imagem de cada caractere para esta faixa e depois para a faixa de -1 a 1, tornando a imagem compatível com os valores esperados pela a próxima etapa do processamento, a rede neural.

F. Reconhecimento do caractere

O reconhecimento do caractere é feito com uma Rede Neural Artificial, através de um classificador *Multilayer Perceptron* - MLP [4, p. 289]. Foi definida a arquitetura apresentada na Figura 6, onde cada um dos 1728 pixels da imagem de entrada (48×36) é ligado diretamente aos neurônios de entrada da rede.

A rede foi configurada com uma camada escondida contendo $m = 116$ neurônios. A camada de saída possui 58 neurônios, a mesma quantidade de classes. A função *sigmóide bipolar* foi utilizada como função de ativação.

Figura 6: Arquitetura de rede neural implementada



A rede foi treinada através do algoritmo da retropropagação do erro com 51 fontes diferentes para 58 classes de caracteres definidas pelo conjunto `ABCDE FGHIJK LMNOPQ RSTUV WXYZ ÀÁÉ ÍÓÚ ÇÃÕ ÂÊÔ abcde fghijk lmnopq rstuv wxyz àáé íóú çãõ âêô 01234 56789 !();:",./?`. Foi utilizado o método *Xavier* [5] para inicialização dos pesos da rede neural, e a atualização dos pesos foi executada com *momentum*.

Os pares de letras maiúsculas e minúsculas equivalentes foram agrupadas nas mesmas classes, reduzindo assim o número de neurônios na saída. Isto resulta num reconhecimento que não distingue maiúsculas minúsculas. Por se tratar de uma aplicação de texto para fala, isto não se torna um limitação.

O vetor *target* para cada padrão de treinamento foi definido como um vetor bipolar onde todas as posições do vetor são -1 exceto a do neurônio em que se deseja a ativação para aquele padrão, onde é definido o valor 1. O resultado do reconhecimento de um padrão foi definido como sendo a ordem do neurônio de saída que possuir o maior valor em sua saída.

O treinamento foi executado com cerca de 48 mil padrões. Aproximadamente um décimo dos padrões foram gerado a partir de fontes presentes no computador, como *Arial*, *Times New Roman*, *Palatino*, e suas variações negrito, condensado, leve, etc. O restante dos padrões foram obtidos através da técnica do *aumento de dados* (*data augmentation*), onde diversas variações distorcidas de cada caractere foram produzidas por um software. A Figura 7 apresenta alguns exemplos destas variações.

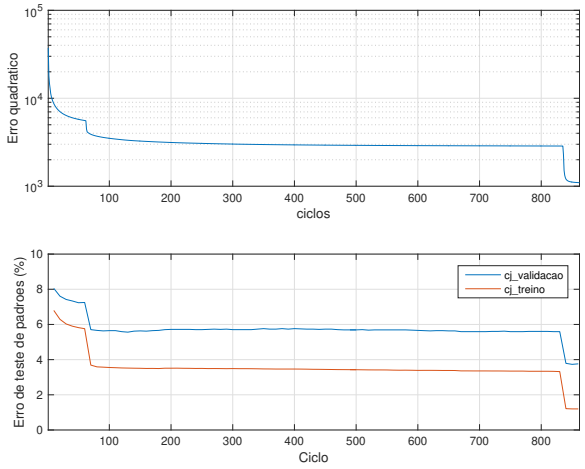
Figura 7: Exemplos de algumas variações aplicadas a um caractere durante a operação de *aumento de dados*

Os padrões foram divididos aleatoriamente em dois conjuntos: treino e validação. O primeiro possui $1/7$ dos total de padrões e o segundo o restante. A rede foi treinada apenas com o conjunto de treino com o objetivo de minimizar o erro quadrático total, mas a cada 10 iterações a rede foi submetida à testes com o conjunto de validação, produzindo um percentual de erro de reconhecimento E_v dos padrões daquele conjunto.

A rede armazena uma cópia separada do seu conjunto de pesos a cada momento que um valor menor de E_v é atingido. No final do treinamento a rede assume esta cópia de pesos. Isto evita o *overfitting*, que ocorre quando a rede se torna muito especializada nos padrões de treino, perdendo a capacidade de generalização.

A Figura 8 apresenta os erros ao longo de 10 horas de treinamento. Observa-se que o erro de reconhecimento no conjunto de validação sempre é maior do que o do conjunto de treino, e em alguns momentos ele chega a aumentar com o passar dos ciclos, enquanto o erro do conjunto de treino sempre é decrescente, assim como o erro quadrático.

Figura 8: Erros ao longo de um treinamento da rede neural



G. Separação em palavras

Os caracteres que são reconhecidos pela rede neural não possuem separação entre as palavras. Esta separação é obtida através de dados do *segmentador de caracteres*, que obtém a distância, em pixels, entre cada caractere segmentado.

Se a distância do caractere atual ao anterior for maior do que a distância média observada em uma determinada janela, assume-se que ali há uma fronteira entre as palavras, devendo ser inserido um caractere de espaço naquela posição, formando assim as palavras reconhecidas.

H. Correção de palavras

Durante toda a cadeia de processamento podem ocorrer alguns erros, como a falha de segmentação entre dois caracteres com espaçamento insuficiente, como os dois caracteres "fo" serem reconhecidos como um caractere só. Outra falha comum é a classificação incorreta da rede neural, como por exemplo do caractere "l" ser reconhecido como o numeral "1" ou mesmo como a letra "I". Assim, as palavras reconhecidas podem apresentar erros de ortografia. Em diversos testes, a palavra "consulta", por exemplo, é reconhecida como "CONSUITA".

É clara a necessidade então de um corretor automático de ortografia das palavras reconhecidas. Neste trabalho foi utilizado o corretor ortográfico *Hunspell*, disponível em `http://hunspell.github.io/`, o mesmo utilizado por exemplo pela suíte de aplicativos de escritório *Libreoffice*. O corretor ortográfico verifica cada palavra com um dicionário e, caso a palavra não exista no dicionário ele apresenta algumas sugestões de palavras candidatas.

Por simplificação, neste trabalho foi adotada sempre a primeira sugestão do corretor ortográfico. Esta simplificação nem sempre se mostra boa o suficiente. Por exemplo, a palavra "CONSUITA" (versão correta seria "consulta") quando submetida ao *Hunspell* apresenta as seguintes sugestões de correção, na ordem:

(i) CONSUETA; (ii) CONSUNTA; (iii) CONSULTA; (iv) CONSUMTA; (v) CONSTITUTA; (vi) CONSCRITA; (vii) CONSTRITA; (viii) CONSUNTIVA; (ix) CONSTITUA; (x) CONSTITUI; (xi) CONSUMIA.

Pela estratégia adotada a palavra selecionada seria a primeira sugestão, claramente incorreta. Isto ocorre pois o *Hunspell* foi feito para correção interativa de palavras, e não automática (por exemplo o usuário do *Libreoffice* normalmente escolhe a palavra correta manualmente dentre as sugestões). Além disso ele não foi projetado para ser um corretor ortográfico pós-OCR, pois não possui a matriz de probabilidades de erros típicos deste tipo de sistema.

I. Síntese de fala

O texto reconhecido e corrigido pelo corretor ortográfico é então submetido a um sintetizador de fala, a última etapa deste trabalho. Neste trabalho foi adotado como primeira opção um sintetizador de código aberto de terceiros, o *eSpeak NG* (`https://github.com/espeak-ng/espeak-ng`). Este

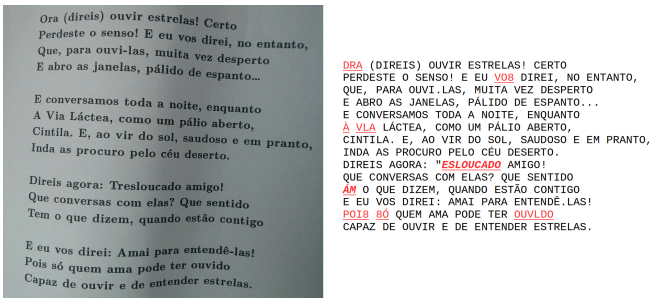
funciona completamente *offline* e possui suporte ao idioma português brasileiro.

As vozes do *eSpeak NG* são robotizadas e em alguns casos não são inteligíveis. Há alternativas utilizando a nuvem (limitadas) como por exemplo a do Google, que apresenta inteligibilidade e qualidade sonora muito superiores. Como alternativa, o software desenvolvido neste trabalho permite a opção pelo serviço do Google através do gTTS (`https://github.com/pndurette/gTTS`), porém o atraso entre a entrada do texto e o início da fala é significativo.

III. CONCLUSÕES E RESULTADOS PRELIMINARES

A Figura 9 apresenta o resultado do processamento de uma fotografia de um texto impresso, com a página propositalmente ondulada.

Figura 9: Resultado de um reconhecimento de texto, sem aplicação do corretor ortográfico



O texto contém 94 palavras. Destas palavras, 9 foram reconhecidas incorretamente (9,6%), sendo que 7 destes erros foram causados por erro de classificação pela rede neural (7,4%) e 2 por erro de segmentação dos caracteres (2,1%).

O reconhecimento de texto se mostrou uma tarefa complexa que envolve diversas etapas distintas de processamento, com grande número de operações de pré e pós processamento.

A segmentação de linhas e de caracteres apresentaram dificuldades consideráveis quando o espaçamento era insuficiente. Muitas fontes fazem a *ligadura tipográfica* entre algumas sequências de caracteres, como entre "f" e "i", com o objetivo de serem mais agradáveis de ler, mas isto confunde o segmentador de caracteres por dificultar o estabelecimento de uma fronteira.

A etapa de treinamento foi um desafio importante, pois com o hardware disponível, foram necessárias cerca de 10 horas por treino para se obter resultados aceitáveis. Levando em conta a experimentação de parâmetros (como a taxa de aprendizagem, número de neurônios na camada escondida), são necessários alguns dias para se ter uma rede neural deste porte treinada.

O corretor ortográfico *Hunspell* se mostrou inapto para a correção das palavras com erros, pois este foi desenvolvido especificamente para corrigir erros de digitação, que possuem uma estrutura distinta dos erros típicos de um sistema OCR.

Como trabalho futuro, é sugerido as seguintes melhorias: mudança da rede neural convencional para uma rede convolucional; melhorias na segmentação de caracteres, especialmente para lidar com as ligaduras tipográficas; detecção e correção de rotação de página; e implementação de um corretor ortográfico próprio para lidar com OCR.

REFERÊNCIAS

[1] SINGH, A.; BACCHUWAR, K.; BHASIN, A. A survey of OCR applications. *International Journal of Machine Learning and Computing*, EJournal Publishing, v. 2, n. 3, p. 314–318, jun 2012. Disponível em: <https://doi.org/10.7763/IJMLC.2012.V2.137>.

[2] SANTOS, J. P. et al. Tecnologia assistiva: um estudo sobre o uso de aplicativos para deficientes visuais. *Brasil Para Todos - Revista Internacional*, v. 4, n. 1, p. 59–69, 2017.

[3] BRADLEY, D.; ROTH, G. Adaptive thresholding using the integral image. *Journal of Graphics Tools*, Informa UK Limited, v. 12, n. 2, p. 13–21, jan 2007. Disponível em: <https://doi.org/10.1080/2151237x.2007.10129236>.

[4] FAUSETT, L. *Fundamentals of neural networks : architectures, algorithms, and applications*. Englewood Cliffs, NJ, EUA: Prentice-Hall, 1994. ISBN 978-0133341867.

[5] GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: TEH, Y. W.; TITTERINGTON, M. (Ed.). *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010. (Proceedings of Machine Learning Research, v. 9), p. 249–256. Disponível em: <http://proceedings.mlr.press/v9/glorot10a.html>.

[6] GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. 3. ed. Upper Saddle River, NJ, EUA: Pearson, 2007. ISBN 978-0131687288.