Revista de Sistemas de Informação da FSMA
n. 22 (2018) pp. 49-59

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

Uma Introdução Compreensiva às Redes Neurais Convolucionais: Um Estudo de Caso para Reconhecimento de Caracteres Alfabéticos

Elivelto Ebermam, Universidade Federal do Espírito Santo, Vitória - ES e
Renato A. Krohling, Universidade Federal do Espírito Santo, Vitória - ES

***Resumo*—As redes Neurais convolucionais tem atraído grande atenção na resolução de tarefas complexas, principalmente em reconhecimento de imagens. Elas foram projetadas especialmente para receber imagens como entradas, pois atuam sobre campos receptivos locais, realizando o processo de convolução. Entretanto, entender o funcionamento dessa rede, pode não ser uma tarefa fácil, principalmente para iniciantes na área de inteligência computacional. Por isso, o objetivo deste trabalho é apresentar de forma didática e intuitiva as redes neurais convolucionais. Um estudo de caso envolvendo reconhecimento de caracteres alfabéticos manuscritos é apresentado para mostrar a viabilidade da abordagem.**

***Palavras-chave*—Redes neurais convolucionais, reconhecimento de caracteres.**

A Comprehensive Introduction to Neural Convolutional Networks: A Case Study for Character Recognition

***Abstract*—Convolutional neural networks have been attracted great attention in the realm of complex tasks, mainly in image recognition. They were specifically designed to process images as inputs, as they act in local receptive fields, performing a convolution process. However, understanding the work principle of this network may not be an easy task, especially for beginners in the area of computational intelligence. Therefore, the objective of this work is to present in a didactic and intuitive way the convolutional neural networks. A case study involving alphabet character recognition is presented in order to ilustrate the feasibility of the approach.**

***Index Terms*—Convolutional neural networks, character recognition.**

I. Introdução

AS redes Neurais artificiais vem ganhando destaque em várias áreas do conhecimento, como reconhecimento de padrões, reconhecimento de caracteres, previsão de séries temporais, entre outros [1]. Desde o seu surgimento até a atualidade, elas foram aperfeiçoadas, adaptando-se a diversas aplicações. Uma delas é a classificação de imagens, no qual a rede neural mais apropriada é a convolucional (do inglês: *convolutional neural network*, abreviada por CNN).

Uma rede neural convolucional [2] possui uma estrutura construída especialmente para receber imagens como entrada. Ela consegue preservar as correlações entre *pixels* vizinhos da imagem, pois atua sobre campos receptivos locais realizando o processo de convolução. Dessa forma, consegue-se reduzir a sensibilidade à translação, rotação e distorção das imagens [3]. Outros tipos de redes neurais não conseguem capturar esse tipo de relação, pois tratam as imagens como uma entrada unidimensional.

As redes neurais convolucionais, porém, são mais complexas que outros tipos de redes neurais, o que dificulta o entendimento do seu funcionamento por iniciantes da área de inteligência computacional.

Embora existam vários artigos sobre redes neurais convolucionais na literatura, há poucos de natureza introdutória ao assunto. O'Shea e Nash [4] fazem uma breve introdução às CNN, discutindo artigos e técnicas recentes para o desenvolvimento das mesmas. Wu [5] discute CNNs do ponto de vista matemático de forma explicativa. No Brasil, tanto quanto é do conhecimento dos autores, existem ainda poucos artigos introdutórios na área. Araújo et al. [6] descrevem os principais conceitos e componentes de CNN, além de apresentar um modelo de forma prática.

Quanto a aplicação, o reconhecimento de caracteres é um problema bem conhecido na literatura. LeCun et al. [7] aplicaram CNNs para uma base de dígitos manuscritos, conhecida como MNIST. Para a mesma base, Mohebi e Bagirov [8] utilizaram mapas auto-organizados (do inglês: *self-organizing maps*, abreviado por SOM) modificado. Bai et al. [9] utilizam uma variação de CNN para reconhecimento de caracteres de diferentes linguagens. Além disso, o reconhecimento de caracteres pode ser utilizado em vários tipos de aplicações, como o reconhecimento automático de placas de veículos por exemplo. Outros exemplos de aplicação incluem digitalização de documentos do setor jurídico, imagem de recibos, processamento de cheques, processamento de formulários de serviços médicos, reconhecimento de CAPTCHA, entre outros [10].

O objetivo deste artigo é apresentar, de forma didática e intuitiva, as redes neurais convolucionais. Para isso

será feita uma introdução sobre neurônio artificial e rede neural tradicional (chamada de *perceptron* multicamadas) na seção 2, pois ela é utilizada na parte final da rede convolucional. A seção 3 discute em detalhes os elementos da rede neural convolucional, focando principalmente nos processos de convolução e *pooling*, os quais são os diferenciais dessa rede. Para melhor assimilação, será apresentado um estudo de caso para reconhecimento de caracteres alfabéticos na seção 4. Por último, são apresentadas as conclusões na seção 5.

II. Redes Neurais Artificiais

UMA rede neural artificial é uma estrutura de processamento de informações paralela e distribuída que consiste em processar elementos interconectados por meio de sinais unidirecionais chamados conexões [11]. Ela apresenta características semelhantes à rede neural biológica, como processamento paralelo e capacidade de aprendizado [12].

As redes neurais artificiais são formadas pela conexão de vários processadores simples, chamados de neurônios artificiais, cada um produzindo uma sequência de ativações de valores reais [13]. O neurônio artificial é a unidade básica de processamento de uma rede neural artificial. Ele se baseia no funcionamento do neurônio natural. Vários modelos foram propostos, porém, o mais utilizado é o baseado na ideia de McCulloch e Pitts [14].

O neurônio artificial é composto basicamente por sinais de entrada, pesos, função de ativação e sinal de saída como ilustrado na Figura 1. Cada entrada x_i é multiplicada por um peso w_i e seus valores são somados. É acrescentado a essa soma um deslocamento linear conhecido como *bias* b, e assim é gerado um sinal de ativação u (calculado por $u = \sum_i x_i \cdot w_i + b$). Esse sinal passa por uma função de ativação $f(u)$ resultando na saída y.

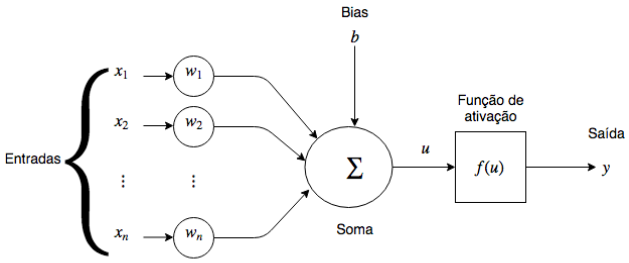

Fig. 1. Neurônio artificial, baseado em [15]

As funções de ativação mais utilizadas são: sigmoidal logística, tangente hiperbólica e unidade linear retificada (do inglês: *Rectified Linear Unit*, abreviada por ReLU). Essas funções são calculadas pelas Equações 1, 2 e 3 respectivamente.

$$f(u) = \frac{1}{1+e^{-u}} \tag{1}$$

$$f(u) = \frac{1-e^{-u}}{1+e^{-u}} \tag{2}$$

$$f(u) = max(0, u) \tag{3}$$

Podemos facilitar a compreensão do processo usando um exemplo. Dado um neurônio com função de ativação sigmoidal logística, com entradas $[x_1 = 0,\ x_2 = 0.5,\ x_3 = 1]$ e pesos $[w_1 = 0.1,\ w_2 = 0.2,\ w_3 = 0.7]$ e $b = -0.8$, gera-se uma saída 0.5, conforme mostrado a seguir:

$$\begin{aligned} u &= \sum_i x_i \cdot w_i + b \\ u &= x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + b \\ u &= 0 \cdot 0.1 + 0.5 \cdot 0.2 + 1 \cdot 0.7 - 0.8 \\ u &= 0 \\ y &= f(u) \\ y &= \frac{1}{1+e^{-u}} \\ y &= \frac{1}{1+e^{-0}} \\ y &= \frac{1}{1+1} \\ y &= 0.5 \end{aligned} \tag{4}$$

Vários neurônios conectados formam uma rede neural artificial, onde a saída y de um neurônio se torna a entrada de outros neurônios no nível seguinte. É comum que eles sejam agrupados em camadas e a ligação entre eles seja unidirecional, com alimentação para frente (*feedforward*). A estrutura de uma rede neural é ilustrada na Figura 2.

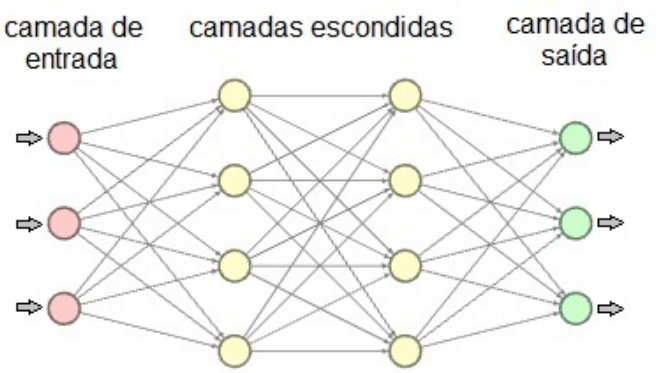

Fig. 2. Estrutura de uma rede neural artificial multicamadas

A primeira camada é a de entrada, a qual apenas recebe os dados e os transmite para a camada seguinte. As camadas intermediárias são chamadas de camadas escondidas, sendo responsáveis pelo processamento das informações. Por último encontra-se a camada de saída, que tem como objetivo apresentar a resposta da rede ao problema em questão, que pode ser a identificação da classe a qual pertence a entrada ou um resultado de processamento em valor contínuo.

III. Redes Neurais Convolucionais

A. Introdução

As redes neurais convolucionais são um tipo de rede neural artificial inspirada no processo biológico no córtex

visual dos animais [16]. As camadas iniciais são especializadas em extrair características dos dados (especialmente imagens), sendo que os neurônios não são totalmente conectados com os da camada seguinte. As camadas finais são totalmente conectadas como na Figura 2 e são responsáveis por interpretar as informações extraídas pelas camadas iniciais e gerar uma resposta. As redes neurais convolucionais tem como precursor o trabalho de Hubel e Wiesel [17], que estudaram o campo visual de gatos, identificando que as células visuais respondem a diferentes tipos de estímulos.

No ano de 1980, Fukushima introduziu uma rede neural chamada Neocognitron. Essa rede é auto-organizada por aprendizado não-supervisionado (sem a necessidade de um tutor) e adquire a capacidade de reconhecer padrões de estímulos baseados na similaridade geométrica [18].

Essa rede possui uma estrutura multi-camadas com níveis de hierarquias, sendo composta por camadas de células simples, complexas e hiper-complexas. Essas células respondem a determinados tipos de estímulos de campos receptivos. A cada nível, a complexidade das características extraídas dos estímulos aumenta, assemelhando-se ao sistema nervoso visual.

Em 1986, houve um grande avanço em redes neurais artificiais com o trabalho de Rumelhart et al. [19], que desenvolveu o algoritmo de aprendizado supervisionado (com a presença de tutor) conhecido como *backpropagation*, que deu a capacidade das mesmas resolverem problemas não-lineares.

Em 1989, LeCun [2] utilizou redes neurais de estrutura hierárquica com detectores de invariância ao deslocamento, chamadas de "*Multilayer Constrained Networks*". Nessas redes as conexões eram feitas de forma local com utilização de pesos compartilhados e somente na última camada eram totalmente conectadas. O treinamento era feito de forma supervisionada por meio de uma variação do algoritmo de aprendizado *backpropagation*.

Em 1993, LeCun et al. [3] referenciaram as redes citadas anteriormente como "*MultiLayer Convolutional Neural Network*" ou simplesmente "*Convolutional Neural Network*". Em 1994, o termo "Convolutional Neural Networks" apareceu no título de um artigo de LeCun et al. [20] e em 1995, LeCun e Bengio [21] publicaram um artigo dedicado à redes neurais convolucionais e suas aplicações.

Em 1998, LeCun et al. [7] descreveram um modelo de rede neural convolucional com um número grande de camadas (sete ao total) chamado LeNet-5.

Em 2012, Krizhevsky et al. [22] desenvolveram um modelo de CNN chamada AlexNet, constituída por 8 camadas. Eles aplicaram o modelo para classificar uma base de dados de larga escala, contendo milhões de imagens, denominada ImageNet. Para realizar o treinamento da rede, utilizaram computação em placas gráficas.

Desde então, com o aumento do poder computacional, os modelos de redes neurais Convolucionais tenderam a crescer em número de camadas, chegando a dezenas e até mesmo centenas de camadas.

B. Arquitetura de uma rede neural convolucional

As redes neurais convolucionais são redes neurais *feedforward* com arquitetura projetada para minimizar a sensibilidade à translação, rotação e distorção da imagem de entrada [3]. Elas são constituídas de: conexões locais, pesos compartilhados, *pooling* e uso de muitas camadas [23].

A rede é organizada em camadas com diferentes funções. A camada convolutiva é responsável por extrair características da imagem, a camada de *pooling* por realizar uma sub-amostragem da imagem e as camadas totalmente conectadas pela interpretação das características extraídas.

C. Camada Convolutiva

Na camada convolutiva, as unidades são organizadas em "*feature maps*", onde cada unidade está conectada a uma parte da camada anterior por meio de um conjunto de pesos chamados "filtros" [23]. As unidades de processamento ou neurônios utilizam uma técnica chamada "pesos compartilhados", a qual consiste em várias conexões sendo controladas por um mesmo parâmetro (peso) [2]. Essa técnica reduz significativamente o número de parâmetros da rede. Além disso, esse tipo de organização de conexões simula o processo de convolução, o qual visa extrair algumas características das imagens, como linhas e contornos por exemplo.

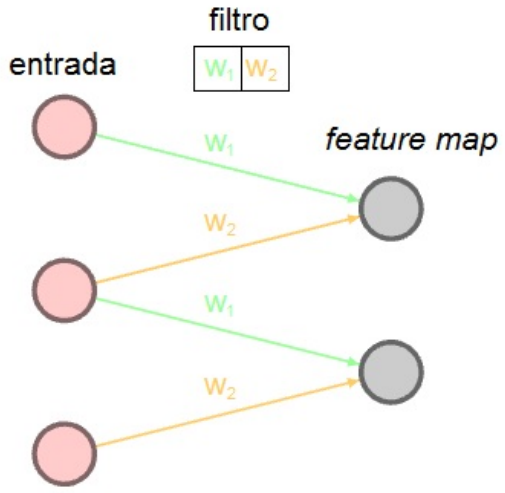

Fig. 3. Compartilhamento de pesos dos neurônios na camada convolutiva

Na Figura 3 é ilustrada a técnica de compartilhamento de pesos. O objetivo do exemplo é mostrar como são feitas as conexões entre a entrada e os neurônios na camada convolutiva (a saída desses neurônios formam os *feature maps*). Cada neurônio (em cinza) se conecta a duas entradas por meio do filtro de pesos [w_1, w_2]. De forma sequencial, pode-se imaginar que no primeiro momento, os dois pesos se ligam as duas primeiras entradas e assim geram a primeira unidade do *feature map*. No segundo momento, os pesos deslizam para baixo, e se conectam à

segunda e terceira entrada formando a segunda unidade do *feature map*. Mais a frente será ilustrado um caso de uma entrada 2D e um filtro 2D.

Os filtros também podem ser organizados em duas ou três dimensões. A cada momento, eles se conectam a uma determinada parte de uma matriz, chamada de campo receptivo local. Nesse caso, os filtros podem ser entendidos como pequenos quadrados, ou cubóides, que deslizam dentro de uma matriz. Eles tem um determinado tamanho, como 5x5 ou 5x5x3 (para entradas em três dimensões) e um passo do deslocamento chamado de "*stride*".

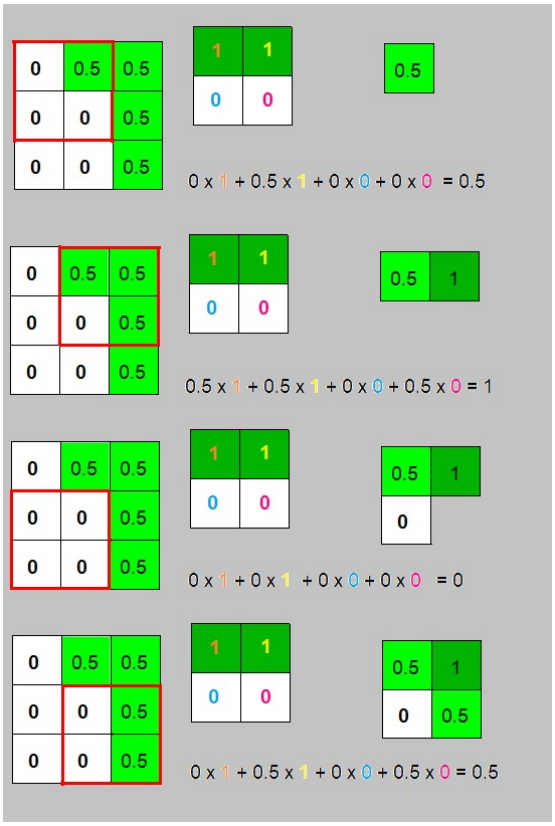

Fig. 4. exemplo de convolução

Na Figura 4 temos uma imagem de entrada 3x3 com valores entre 0 (zero) e 1 (um), onde 0 (zero) são *pixels* mais claros e 1 (um) mais escuros. O filtro utilizado tem tamanho 2x2, tendo os pesos [1,1,0,0] e atua sobre campos receptivos locais (quadrado com bordas vermelhas) da imagem de mesmo tamanho (2x2). Cada peso do filtro é multiplicado pelo valor do *pixel* correspondente no campo receptivo. Esses valores são somados, adicionado o termo *bias* e assim gera-se a saída da unidade correspondente no *feature map*. Nesse exemplo, o *bias* não é mostrado na figura e, portanto, seu valor deve ser considerado como 0. O tamanho do *stride* utilizado é 1 e isso significa que, a cada momento, o campo receptivo local desloca-se um *pixel* para a direita. Quando chega-se ao final de uma linha (como no segundo momento na imagem), desloca-se um *pixel* para baixo e volta-se para o início da linha.

Como mostrado na Figura 4, a aplicação de um filtro (2x2) em uma imagem (3x3) resulta em um *feature map* (2x2). O cálculo do tamanho do *feature map* é feito conforme:

$$T_m = \frac{M-k}{s} + 1 \tag{5}$$

onde T_m é o tamanho da *feature map*, M é o tamanho da imagem, k é o tamanho do filtro (*kernel*) e s é o passo do deslocamento (*stride*). As variáveis t_m, M e k representam o tamanho em uma das dimensões (horizontal ou vertical), porém é comum a utilização de imagens e filtros quadrados (altura e largura iguais). Assim as variáveis representam os tamanhos em ambas as dimensões.

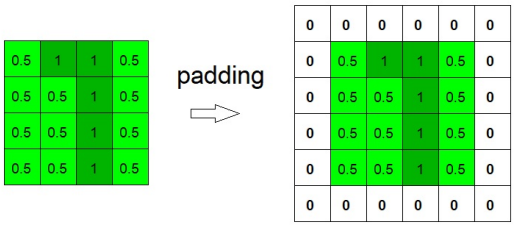

Fig. 5. Ilustração de um exemplo de zero-padding

Para se conseguir um tamanho de *feature map* específico, às vezes é necessário alterar o tamanho da imagem de entrada. Isso é feito inserindo-se uma borda chamada *padding* ou *zero-padding*, que consiste em inserir zeros em torno da imagem, como ilustrado na Figura 5. Dessa forma, o tamanho resultante dos *feature maps* passa a considerar a espessura da borda p e nesse caso então, será calculado por:

$$T_m = \frac{M+2p-k}{s} + 1 \tag{6}$$

Após gerar os valores do *feature map*, é necessário passá-los por uma função de ativação. Isso é feito para que sejam capazes de resolver problemas não-lineares (no caso a função de ativação precisa ser não-linear também). Comumente a mais usada em redes neurais convolucionais é a ReLU.

No algoritmo 1 está resumido o processo realizado pela camada de convolução.

Algorithm 1 convolução

1: Inicialize os valores dos pesos w com valores pequenos
2: **para** $i = 1$ até $(M-k)/s+1$ **faça**
3: **para** $j = 1$ até $(N-k)/s+1$ **faça**
4: $u_{i,j} = \sum_{c=1}^{k}\sum_{d=1}^{k} x_{i\cdot s-1+c, j\cdot s-1+d} \cdot w_{c,d} + b$
5: $y_{i,j} = f(u_{i,j})$
6: **fim para**
7: **fim para**

No pseudo-código, $x_{i,j}$ é o valor do *pixel* na posição (i,j) da imagem x, M é a altura e N é a largura da imagem. A variável $u_{i,j}$ indica o valor da posição (i,j) do *feature map* e $y_{i,j}$ é a saída da mesma após passar pela função de ativação. A variável $w_{c,d}$ indica o peso do filtro na posição (c,d).

D. Camada de pooling

A camada de *pooling* tem como objetivo alcançar certo nível de invariância ao deslocamento, reduzindo a resolução dos *feature maps* [24]. Ela atua de forma parecida com uma camada convolutiva, porém, o "*stride*" normalmente é o mesmo que o tamanho do filtro (ao contrário da camada anterior onde o *stride* era 1). Isso faz com que os campos receptivos sejam totalmente distintos e que a redução dos *features maps* seja de 75%. Os tipos mais comuns de *pooling* são: *max pooling* e *average pooling*. No *max pooling* seleciona-se o maior valor do campo receptivo e no *average pooling* calcula-se a média dos valores.

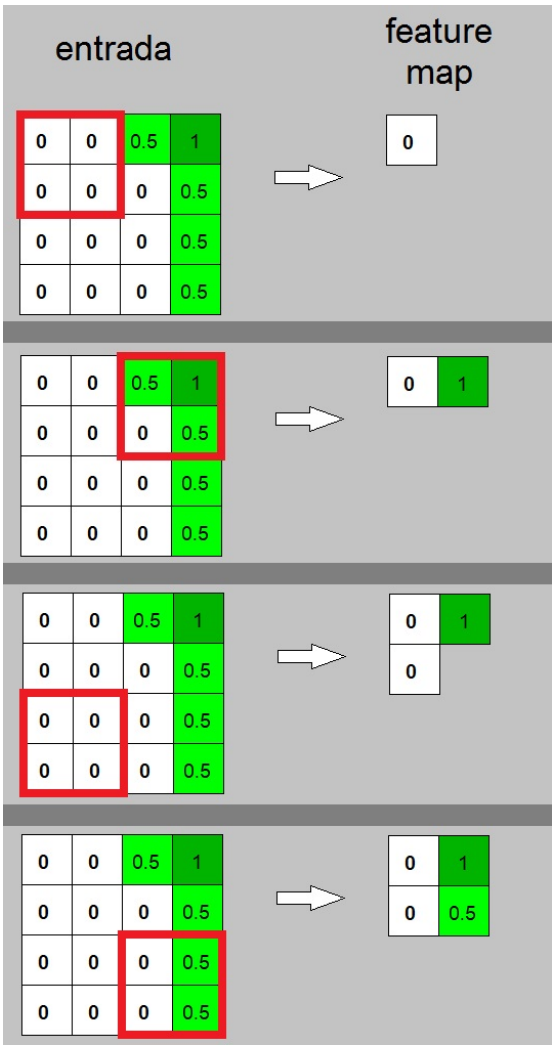

Fig. 6. Ilustração de um processo de max pooling

Na Figura 6, a camada de *pooling* recebe como entrada uma imagem em tons de verde e realiza a operação de *max pooling* sobre ela. Como resultado, a saída é uma imagem com a metade da altura e metade da largura original.

O processo de *pooling* realizado após uma camada de convolução é descrito no algoritmo 2 (considerando $s = k$).

Algorithm 2 *pooling*

para $i = 1$ até $(M - k)/s + 1$ **faça**
  para $j = 1$ até $(N - k)/s + 1$ **faça**
    se *max pooling* **então**
      $y_{i,j} = max(x_{(i-1)\cdot k+1,(i-1)\cdot k+1}, \ldots, x_{i\cdot k,j\cdot k})$
    fim se
    se *average pooling* **então**
      $y_{i,j} = (\sum_{c=1}^{k} \sum_{d=1}^{k} x_{i\cdot s-1+c,j\cdot s-1+d})/(k^2)$
    fim se
  fim para
fim para

E. Camada totalmente conectada

A entrada passa por camadas convolutivas e de *pooling*, que identificam das características mais simples às mais complexas até chegarem as camadas totalmente conectadas. Normalmente, a camada anterior é uma de *pooling*, onde seus *feature maps* apresentam mais de uma dimensão, e por isso são redimensionadas para uma só (em forma de vetor) para poder se conectar à parte final da rede. As camadas totalmente conectadas funcionam como uma rede neural multicamadas *feedforward* e são responsáveis pela interpretação das características extraídas pelas camadas iniciais.

O pseudo-código do processamento da camada totalmente conectada é dada no algoritmo 3.

Algorithm 3 *Camada totalmente conectada*

se camada anterior é de *pooling* ou de convolução **então**
  redimensionar(x)
fim se
para todo neurônio j da camada totalmente conectada **faça**
  para toda entrada i da camada anterior **faça**
    $u_j = \sum_{i=1}^{n} x_i \cdot w_{i,j} + b_j$
    $y_j = f(u_j)$
  fim para
fim para

F. Softmax

A última camada é responsável por apresentar os resultados. A quantidade de neurônios nela é definida pelo número de classes do problema. Cada neurônio apresenta a saída em forma de probabilidade (0 a 1) e a resposta é o número do neurônio com saída de maior valor. Isso é feito com a função *softmax*, descrita pela equação:

$$y_j = \frac{e^{y(j)}}{\sum_{i=1}^{n} e^{y(i)}} \tag{7}$$

onde a saída y do neurônio j passa a ser o valor da mesma dividido pelo somatório de todas as saídas dos neurônios dessa camada.

Por exemplo, considere que as saídas da rede neural antes da aplicação da função *softmax* sejam: $[y_1 = 1, y_2 = 1.5, y_3 = 2]$. Então as saídas da rede serão calculadas conforme:

$$
\begin{aligned}
y_j &= \frac{e^{y(j)}}{\sum_{i=1}^{n} e^{y(i)}} \\
y_j &= \frac{e^{y_j}}{e^{y_1} + e^{y_2} + e^{y_3}} \\
y_j &= \frac{e^{y_j}}{e^{1} + e^{1.5} + e^{2}} \\
y_j &= \frac{e^{y_j}}{2.72 + 4.48 + 7.39} \\
y_j &= \frac{e^{y_j}}{14.59} \\
y_1 &= \frac{e^{y_1}}{14.59} = \frac{e^{1}}{14.59} = \frac{2.72}{14.59} = 0.186 \\
y_2 &= \frac{e^{y_2}}{14.59} = \frac{e^{1.5}}{14.59} = \frac{4.48}{14.59} = 0.307 \\
y_3 &= \frac{e^{y_3}}{14.59} = \frac{e^{2}}{14.59} = \frac{7.39}{14.59} = 0.507
\end{aligned}
\tag{8}
$$

Assim, completa-se a construção da rede neural convolucional, a qual é ilustrada na Figura 7.

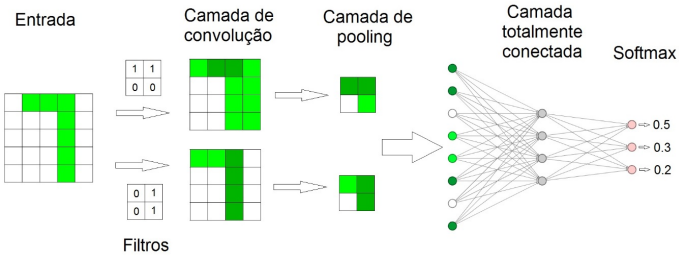

Fig. 7. Rede neural convolucional

G. Aprendizado

Por padrão, o aprendizado é feito pelo método de retropropagação do erro, no qual usa a descida do gradiente para atualizar os pesos [24]. Para calcular o erro é necessário usar uma determinada medida ou função de custo, como a distância Euclidiana entre a resposta da rede e o valor esperado. Para classificação porém, quando há mais de duas classes, normalmente se usa a função de entropia cruzada descrita por:

$$C = -\sum_{i=1} y_i log(\hat{y}_i) \tag{9}$$

onde C é o custo, y_i é a saída de um neurônio e $\hat{y}_i$ é a saída desejada.

Para treinar a rede, deseja-se minimizar a função de custo, alterando-se os pesos. A atualização é feita conforme:

$$\Delta w_i^t = \eta \frac{\partial C}{\partial w_i^t} \tag{10}$$

$$w_i^{t+1} = w_i^t - \Delta w_i^t \tag{11}$$

onde w_i^t é um peso i da rede no instante (t) atual, η é a taxa de aprendizagem e Δw_i^t é a variação do peso i no instante atual.

Para suavizar a descida do gradiente, aproveitando parte de seu movimento anterior é utilizado o termo *momentum*, conforme:

$$w_i^{t+1} = w_i^t - \Delta w_i^t + \alpha \Delta w_i^{t-1} \tag{12}$$

onde α é o *momentum* e seu valor pondera a contribuição da variação dos pesos feita anteriormente.

Além disso, a descida de gradiente pode ser feita de forma estocástica, por meio do método gradiente descendente estocástico (do inglês: *stochastic gradient descent*, abreviado por SGD). Esse método estima os gradientes com base em um único exemplo selecionado aleatoriamente [24]. Na prática, contudo, os gradientes são calculados com base em um conjunto de exemplos, denominado *mini-batch* [24]. O uso de *batches* requer menos atualizações de pesos que o modo incremental (um exemplo por vez), e isso reduz o tempo de treinamento (visto que pode ser custoso, por exemplo, quando há muitas imagens). As atualizações também se tornam mais precisas.

A cada seleção aplica-se o processo de atualização dos pesos da rede. Isso é feito até completar todo o conjunto de amostras. A esse processo de ajustar os pesos para todas as amostras de treino, dá-se o nome de época.

Alguns outros algoritmos de treinamento baseados em descida de gradiente são: RMSProp, Adagrad, Adadelta, Adam, entre outros.

H. Dropout

Às vezes, pode ocorrer que a rede aprenda classificar muito bem as imagens de seu conjunto de treinamento, porém, em outras imagens, ela apresente um desempenho ruim. Para evitar que ocorra esse sobreajuste (em inglês *overfitting*) em seu aprendizado, é utilizada a técnica de *dropout*. Ela consiste em retirar aleatoriamente algumas unidades da rede neural [25], como ilustrado na Figura 8. É definida uma probabilidade p, onde cada unidade tem essa probabilidade de seu sinal ser propagado na rede.

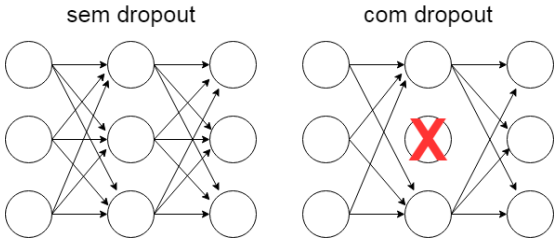

Fig. 8. Dropout. Na direita temos uma rede neural sem dropout, e portanto com todos o neurônios ativos. Já à direita, um neurônio da segunda camada é aleatoriamente desligado, caracterizando o uso de dropout.

I. Plataformas para desenvolvimento

Existem várias ferramentas que já trazem consigo a implementação de vários mecanismos que auxiliam na construção de redes neurais convolucionais, como convolução,

pooling, funções de ativação, algoritmos de aprendizagem entre outros.

TensorFlow [26] é uma biblioteca de aprendizado de máquina em larga escala, desenvolvido por pesquisadores da empresa Google. Ela normalmente é utilizada por meio da linguagem de programação Python, através de uma API. Outras linguagens também são suportadas como C++ e Java.

Keras [27] é uma API de alto nível para construção de redes neurais, no qual atua sobre outros *frameworks* como TensorFlow, CNTK ou Theano.

Caffe [28] é um *framework* de código aberto desenvolvido pela *Berkeley Vision and Learning Center* (BVLC). Seu código foi escrito em C++, usando CUDA para computação em placa gráfica, sendo que também fornece conexões para Python/Numpy e MATLAB.

Torch [29] é um *framework* de computação científica com suporte a vários algoritmos de aprendizado de máquina. Ele é utilizado através de uma linguagem de *script* chamada LuaJIT.

Essas ferramentas tornam o processo de construção de uma rede neural mais rápido e simplificado. Além disso, através delas, o usuário pode aumentar o desempenho do treinamento da rede neural, por meio do processamento em placas gráficas, sem nenhum código adicional.

Neste trabalho, foi escolhido o framework Tensorflow pois ele utiliza a linguagem Python, que é bem intuitiva. Além disso, possui grande flexibilidade, ou seja, permite elaborar e modificar várias estruturas de redes neurais.

IV. Máquina de aprendizado extremo

A máquina de aprendizado extremo (ELM) é um algoritmo de aprendizagem para redes neurais artificiais com uma única camada escondida [32]. Os pesos entre a camada inicial e a camada escondida são iniciados aleatoriamente. No entanto, diferentemente de algumas outras redes neurais, onde esses pesos são modificados iterativamente, na ELM eles não são modificados. O treinamento da rede ocorre na matriz de pesos entre a camada escondida e a camada de saída, a qual é calculada analiticamente. Por não requerer um processo iterativo, geralmente o treinamento da rede é feito de forma muito rápida.

A primeira etapa do algoritmo ELM consiste em formar a matriz H, que é a matriz resultante da saída dos neurônios da camada escondida para cada umas das amostras de entrada, conforme:

$$H = \begin{bmatrix} f(x_1 w_1 + b_i) & \dots & f(x_1 w_n + b_m) \\ \vdots & \ddots & \vdots \\ f(x_m w_1 + b_i) & \dots & f(x_m w_n + b_n) \end{bmatrix} \quad (13)$$

onde as linhas representam as amostras e as colunas representam os neurônios da camada escondida, m é a quantidade de amostras de entrada e n é a quantidade de neurônios na camada escondida. Nesse caso, x_i é vetor de atributos e w_i é um vetor de pesos.

A segunda etapa consiste em calcular $\hat{\beta}$, que é a matriz de pesos entre a camada escondida e a camada de saída. Ela é calculada como a solução dos mínimos quadrados para o sistema linear $H\hat{\beta} = T$, sendo $\hat{\beta} = H^{\dagger}T$, onde $H^{\dagger}$ é a inversa generalizada de Moore-Penrose da matriz H e T é o vetor de respostas para as amostras de entrada [32], [33].

Considere o problema da porta lógica AND onde há dois tipos de entrada: 1 (passagem de corrente elétrica) e 0 (sem passagem de corrente elétrica). A porta lógica recebe duas entradas e retorna uma resposta: $0\&0 = 0$, $0\&1 = 0$, $1\&0 = 0$ e $1\&1 = 1$. Assim, X e T são definidos conforme:

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad T = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (14)$$

Considere também que a ELM tenha dois neurônios na camada escondida com função de ativação sigmoidal logística, e que os pesos W e os *bias* B foram definidos aleatoriamente conforme:

$$W = \begin{bmatrix} 0.2 & 0.5 \\ 0.6 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} -0.5 & 0.5 \end{bmatrix} \quad (15)$$

Assim $\hat{\beta}$ é calculado conforme:

$$\hat{\beta} = H^{\dagger}T$$

$$\hat{\beta} = (f(XW + B))^{\dagger}T$$

$$\hat{\beta} = \begin{bmatrix} f(x_{11}\dot{w}_{11} + x_{12}\dot{w}_{21} + b_1) & f(x_{11}\dot{w}_{21} + x_{12}\dot{w}_{22} + b_2) \\ f(x_{21}\dot{w}_{11} + x_{22}\dot{w}_{21} + b_1) & f(x_{21}\dot{w}_{21} + x_{22}\dot{w}_{22} + b_2) \\ f(x_{31}\dot{w}_{11} + x_{32}\dot{w}_{21} + b_1) & f(x_{31}\dot{w}_{21} + x_{32}\dot{w}_{22} + b_2) \\ f(x_{41}\dot{w}_{11} + x_{42}\dot{w}_{21} + b_1) & f(x_{41}\dot{w}_{21} + x_{42}\dot{w}_{22} + b_2) \end{bmatrix}^{\dagger} T$$

$$\hat{\beta} = \begin{bmatrix} f(0 \cdot 0.2 + 0 \cdot 0.6 + (-0.5)) & f(0 \cdot 0.5 + 0 \cdot 1 + 0.5) \\ f(0 \cdot 0.2 + 1 \cdot 0.6 + (-0.5)) & f(0 \cdot 0.5 + 1 \cdot 1 + 0.5) \\ f(1 \cdot 0.2 + 0 \cdot 0.6 + (-0.5)) & f(1 \cdot 0.5 + 0 \cdot 1 + 0.5) \\ f(1 \cdot 0.2 + 1 \cdot 0.6 + (-0.5)) & f(1 \cdot 0.5 + 1 \cdot 1 + 0.5) \end{bmatrix}^{\dagger} T$$

$$\hat{\beta} = \begin{bmatrix} f(-0.5) & f(0.5) \\ f(0.1) & f(1.5) \\ f(-0.3) & f(1) \\ f(0.3) & f(2) \end{bmatrix}^{\dagger} T$$

$$\hat{\beta} = \begin{bmatrix} \frac{1}{1+e^{-(-0.5)}} & \frac{1}{1+e^{-0.5}} \\ \frac{1}{1+e^{-0.1}} & \frac{1}{1+e^{-1.5}} \\ \frac{1}{1+e^{-(-0.3)}} & \frac{1}{1+e^{-1}} \\ \frac{1}{1+e^{-0.3}} & \frac{1}{1+e^{-2}} \end{bmatrix}^{\dagger} T$$

$$\hat{\beta} = \begin{bmatrix} 0.38 & 0.62 \\ 0.53 & 0.82 \\ 0.43 & 0.73 \\ 0.57 & 0.88 \end{bmatrix}^{\dagger} T$$

$$\hat{\beta} = \begin{bmatrix} -6.4520 & 6.9861 & -17.0639 & 12.2380 \\ 4.3023 & -4.0280 & 10.9916 & -7.2892 \end{bmatrix} T$$

$$\hat{\beta} = \begin{bmatrix} -6.4520 & 6.9861 & -17.0639 & 12.2380 \\ 4.3023 & -4.0280 & 10.9916 & -7.2892 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\hat{\beta} = \begin{bmatrix} 12.2380 \\ -7.2892 \end{bmatrix} \tag{16}$$

O cálculo da inversa generalizada é um pouco mais complexo, porém a maioria das linguagens de programação tem bibliotecas para o cálculo da mesma. Tendo a matriz $\hat{\beta}$, a saída y da ELM para uma amostra de teste Z é dada conforme:

$$y = f(ZW + B) * \hat{\beta} \tag{17}$$

V. Estudo de caso

Embora MNIST seja uma base de dados de reconhecimento de dígitos padrão e largamente utilizado na literatura [30], neste trabalho será utilizado o problema de reconhecimento de caracteres alfabéticos de diversas fontes para ilustrar a aplicação de CNN. Essa escolha foi feita devido ao fato do problema se assemelhar ao primeiro, porém, a rede terá que classificar entre 26 letras em vez de apenas 10 dígitos, o que torna a tarefa um pouco mais complexa.

A. Base de dados

A base de dados "Alfabeto" foi desenvolvida por alunos do laboratório de pesquisa Labcin (Laboratório de Computação e Engenharia Inspirada na Natureza) do programa de pós-graduação em informática. A mesma contem um total de 11.960 imagens de caracteres alfabéticos em escala de cinza, tanto minúsculas quanto maiúsculas, divididas em 26 classes (cada uma correspondente a uma letra). Na Figura 9 estão ilustrados alguns exemplos.

Fig. 9. Exemplos de amostras da base de dados

A base de dados foi dividida em três partes: 70% para treino, 15% para validação e 15% para teste. O valor de cada *pixel* está no intervalo de 0 a 255, porém foram normalizados entre 0 e 1.

B. Configuração da rede neural convolucional

A rede neural convolucional utilizada [1] é apresentada na Figura 10. Ela recebe como entrada uma imagem de 30 por 30 *pixels*, seguida de uma camada convolutiva de 32 filtros (em verde). Cada filtro possui 5x5 pesos, com deslocamento de 1 *pixel* e *padding* de 4 *pixels* para manter o tamanho original da imagem. Assim, são gerados 32 *feature maps* de 30x30 *pixels* (em azul).

Em cada um desses *feature maps* é aplicada a operação de *max pooling* de tamanho 2x2 *pixels*, reduzindo ambas as dimensões de cada um deles pela metade. Como resultado tem-se uma nova imagem de 15x15x32 *pixels* (em laranja), ou seja, uma imagem de 32 canais.

Nessa imagem uma nova operação de convolução é aplicada, tendo 64 filtros com dimensão 5x5x32, com os mesmos valores de deslocamento e *padding* da primeira. Como resultado tem-se 64 *feature maps* de 15x15 *pixels*, sendo que cada um deles passa pela operação de *max pooling*. Com isso passa-se a ter uma imagem de 8x8x64 *pixels*, a qual é vetorizada em apenas uma dimensão com 4096 *pixels*.

Cada *pixel* é entrada da primeira camada totalmente conectada, com 512 neurônios (em cinza). É utilizado *dropout* com probabilidade de 50% do neurônio propagar o sinal. As saídas dos neurônios da primeira camada se ligam com outra de 512 neurônios e novamente é utilizado o mesmo *dropout*.

Por último há uma camada com 26 neurônios (em rosa), o qual é seguida por uma função *softmax*, que determina as probabilidades de cada classe.

Para o treinamento, as imagens foram misturadas e foram utilizados *batches* de 52 imagens com 2 imagens de cada classe em cada. A função de ativação utilizada foi a Rectified Linear Unit (ReLU). O método de aprendizado utilizado foi o gradiente descendente estocástico (SGD) com momentum de 0.9 e a taxa de aprendizagem foi de 0.01.

C. Resultados de simulação

FORAM realizados 30 testes em um computador AMD Phenom X2 com 4 Gigabytes de Memória RAM. O código foi desenvolvido na linguagem Python com auxílio da API TensorFlow. A rede teve acurácia média de 91.08% de acertos, com desvio padrão de 1.44% e o melhor resultado alcançado foi 93.1%. O treinamento da rede levou em média 1761.52 segundos, com desvio padrão de 118.09 segundos.

A Tabela I mostra a matriz de confusão das letras, sendo que na vertical estão as letras corretas e na horizontal as respostas da rede. Uma matriz de confusão, ou matriz de erro, é uma forma eficiente de representar a precisão da classificação, onde descreve claramente a acurácia de cada categoria juntamente com os erros de inclusão e os erros de exclusão presentes [31].

Para a maioria das letras a rede conseguiu uma boa classificação. As letras que a rede teve maior facilidade em reconhecer foram: p (98.5%), w (97.1%), b (95.65%) e t (95.65%). As letras que a rede teve mais dificuldade foram: l (65.21%), i (76.81%), a (85.5%), n (86.95%) e v (86.95%). As principais confusões da rede foram: trocar l por i (28.98%), i por l (13.04%), v por u (5.79%), i por j (4.34%), n por h (4.34%) e v por y (4.34%).

[1] O código fonte pode ser solicitado aos autores via e-mail (elivelt oebermam@gmail.com)

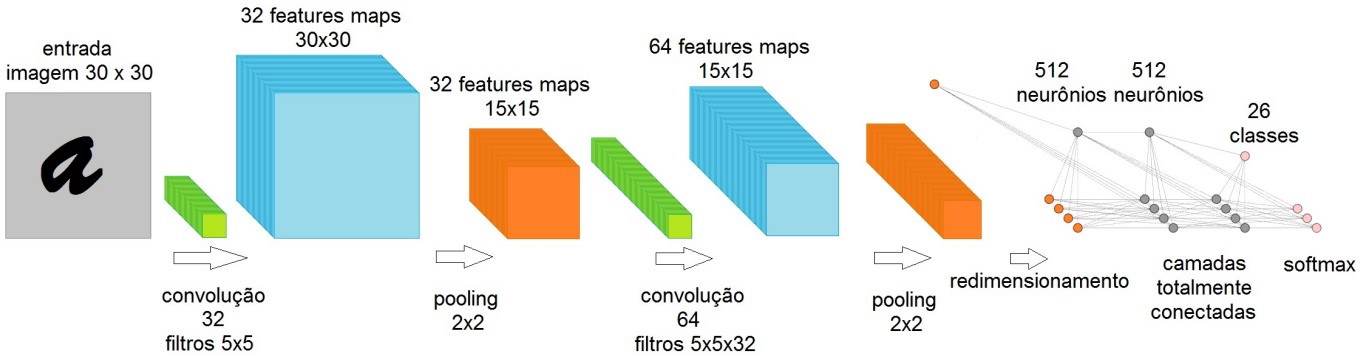

Fig. 10. Arquitetura da rede neural convolucional utilizada

TABELA I
MATRIZ DE CONFUSÃO PARA A BASE DE DADOS DE CARACTERES ALFABÉTICOS

	a	b	c	d	e	f	g	h	i	J	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	59		2				2						1		1			1			2					1
b		66					1														1					1
c	1		65		1										2											
d				63	2			1				1					1				1					
e			1		64		1													1				1		1
f					1	61			2							1		2		1					1	
g	1				2		61										2				1	1				1
h	2							63						1			1	1							1	
i			1						53	3		9						1	1	1						
J		1							2	65											1					
k		2						1			61	1						1			1			2		
l	1				1				20	1		45						1								
m					1			1					65	2												
n		1						3						60				2			1			2		
o		1		2											64		1	1								
p				1												68										
q							2								2		64				1					
r	1										1						1	63	1	1					1	
s	1		2				1			1									64							
t						1												1		66		1				
u									2								1		1		63		2			
v															1						4	60	1		3	
w						1															1		67			
x	1				1												1							63	3	
y							1		1	1							1				2				63	
z	1				2							1														65

Pelo fato da base misturar letras minúsculas com maiúsculas, a possibilidade de haver letras parecidas é maior, como é o caso do "I" (i maiúsculo) com o "l" (L minúsculo). Às vezes, uma letra de determinada fonte é muito difícil de reconhecer ou fácil de confundir isoladamente, devido ao fato de que as pessoas só conseguem distinguí-las quando estão em um contexto.

D. Comparação com outro método

Os resultados da CNN foram comparados ao de outra rede neural muito utilizada atualmente: a máquina de aprendizado extremo (do inglês: *extreme learning machine*, abreviado por ELM) [32], [33]. Essa rede neural é apresentada em detalhes no Apêndice A.

Os experimentos para a ELM foram realizados da mesma forma que para a CNN. A ELM utilizada apresenta 900 unidades de entrada (uma para cada *pixel* da imagem), 3000 neurônios na camada escondida e 26 neurônios na camada de saída. A função de ativação utilizada foi a sigmoidal logística.

A ELM teve uma acurácia média de classificação de 77.03% com desvio padrão de 0.75%. A mesma rede levou em média 51 segundos para realizar o treinamento, com um desvio padrão de 5.88 segundos. As figuras 11 e 12 apresentam em forma de *box plot*, a acurácia da classificação para os dados de teste e o tempo computacional do treinamento respectivamente, da CNN e ELM.

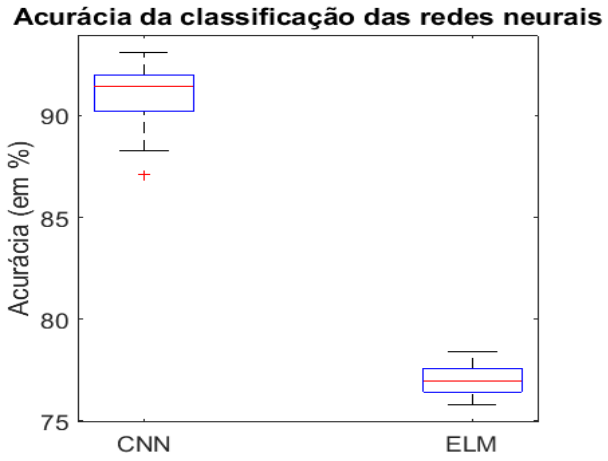

Fig. 11. Acurácia da classificação para os dados de teste da base de caracteres alfabéticos

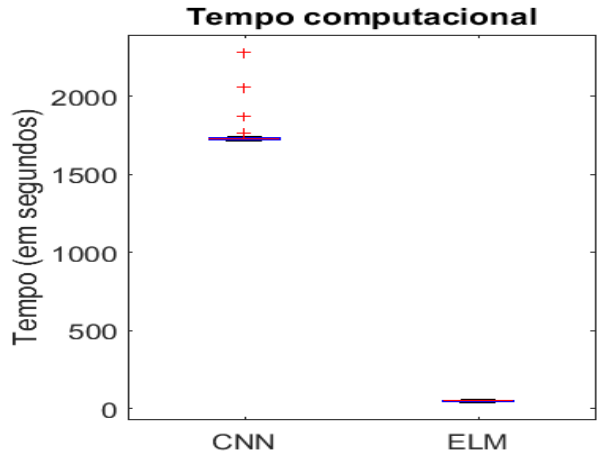

Fig. 12. Tempo computacional para o treinamento da rede neural

E. Avaliação do desempenho da rede neural convolucional

A rede neural convolucional, atuando sobre campos receptivos locais da imagem consegue preservar informações

importantes, como a correlação entre *pixels* vizinhos na imagem. Isso é uma vantagem dela sobre outras redes neurais (como a ELM), as quais precisam redimensionar a imagem para uma dimensão, perdendo assim, a estrutura original da imagem. Dessa forma, como visto no caso de classificação de caracteres alfabéticos, a CNN apresenta melhor acurácia de classificação.

A estrutura da CNN é também eficiente para múltiplas camadas. A técnica de pesos compartilhados e *pooling* acaba reduzindo o número de pesos da rede, facilitando o treinamento dela. Outras redes neurais do tipo *feedforward* apresentam dificuldades no treinamento à medida em que se adicionam camadas.

Entretanto, o tempo de treinamento para a CNN é alto. Para o estudo de caso, ela demorou cerca de 30 vezes mais que a ELM. Mesmo com técnicas para redução de pesos, o número de conexões ainda é alto. Treinar vários pesos de forma iterativa é custoso. Alguns modelos de CNN precisam de um hardware adequado para treiná-las (normalmente placas gráficas de última geração).

VI. Considerações Finais

NESTE trabalho foi apresentada uma introdução compreensiva às redes neurais convolucionais de forma intuitiva e explicativa. Os mecanismos da rede foram explicados detalhadamente, ilustrando os processos. Além disso, foi apresentado um estudo de caso da aplicação de um modelo de CNN para reconhecimento de caracteres alfabéticos manuscritos e foi feita uma comparação com outra rede neural, a ELM. Os resultados de ambas as técnicas foram descritas em termos de média e desvio padrão, para a acurácia da classificação e tempo computacional do treinamento. Os testes foram realizados 30 vezes, sendo apresentados em forma de *box plots*. A acurácia da classificação da CNN foi um pouco baixa, se comparada a casos em que é aplicada a base MNIST, por exemplo. No entanto, a base de dados utilizada no trabalho apresenta alguns casos bem difíceis como 'i' maiúsculo e 'L' minúsculo, que são bem parecidos. Ainda assim, em termos de acurácia, a CNN foi bem superior a ELM. Acredita-se que, modificando a estrutura da rede e outros parâmetros os resultados possam melhorar. Entretanto, o objetivo do trabalho é mostrar a CNN funcionando de forma simples, nesse caso aplicando um modelo padrão, uma LeNet. Com isso espera-se que, iniciantes na área de inteligência computacional, possam utilizar esse material para melhor entendimento das redes neurais convolucionais. Além disso, pode-se ampliar a aplicação da CNN para reconhecimento de palavras e de placas de automóveis por exemplo. Outra expansão é a aplicação para imagens coloridas, como reconhecimento de animais, plantas ou objetos.

Agradecimentos

E. Ebermam agradece ao CNPq pela bolsa de mestrado. R. A. Krohling agradece ao CNPq e a FAPES (Fundação de Amparo à Pesquisa do Espirito Santo) pelo suporte financeiro, concessões No. 309161/2015-0 e No. 039/2016, respectivamente. Os autores agradecem aos alunos Gabriel Giorisatto, André Georghton Cardoso Pacheco, Carlos Alexandre Siqueira da Silva e Igor de Oliveira Nunes pela elaboração e disponibilização da base de dados "Alfabeto". Os autores também agradecem aos revisores e ao editor-chefe da revista pelas sugestões para o aprimoramento do trabalho.

Referências

[1] Z. Bingul, H. M. Ertunc, & C. Oysu. Applying neural network to inverse kinematic problem for 6R robot manipulator with offset wrist. In *Adaptive and Natural Computing Algorithms*, pp. 112-115. Springer, Vienna. 2005.

[2] Y. LeCun, et al., Generalization and network design strategies, *Connectionism in perspective*, 1989, pp.143–155.

[3] Y. Lecun, Y. Bengio, D. Henderson, A. Weisbuch, H. Weissman, L. Jackel. On-line handwriting recognition with neural networks: Spatial representation versus temporal representation, Ecole Nationale Superieure des Telecommunications, 1993.

[4] K. O'Shea, R. Nash, An introduction to convolutional neural networks, arXiv preprint arXiv:1511.08458. 2015.

[5] J. Wu, Introduction to convolutional neural networks, National Key Lab for Novel Software Technology, Nanjing University, China. 2017.

[6] F. H. Araújo, A. C. Carneiro, R. R. Silva, Redes neurais convolucionais com tensorflow: Teoria e prática. III Escola Regional de Informática do Piauí. Livro Anais - Artigos e Minicursos, v. 1, n. 1, 2017, pp. 382-406.

[7] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. In *Proceedings of the IEEE 86* (11), 1998, pp.2278–2324.

[8] E. Mohebi & A. Bagirov. A convolutional recursive modified Self Organizing Map for handwritten digits recognition. *Neural Networks*, vol. 60, 2014, pp. 104-118.

[9] J. Bai, Z. Chen, B. Feng, B. Xu, Image character recognition using deep convolutional neural network learned from different languages, in: *2014 IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 2560–2564.

[10] K. A. Hamad, M. Kaya, A detailed analysis of optical character recognition technology, *International Journal of Applied Mathematics, Electronics and Computers*, vol. 4, Special Issue-1, 2016, pp. 244-249.

[11] R. Hecht-Nielsen, et al., Theory of the backpropagation neural network., *Neural Networks 1 (Supplement-1)*, 1988, pp.445–448.

[12] T. Fukuda, T. Shibata, M. Tokita, T. Mitsuoka. Neural network application for robotic motion control-adaptation and learning, In *IEEE International Joint Conference on Neural Networks*, vol. 2, 1990, pp. 447-451.

[13] J. Schmidhuber, Deep learning in neural networks: An overview, *Neural Networks* 61, 2015, pp. 85–117.

[14] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, *The Bulletin of Mathematical Biophysics* 5 (4), 1943, pp. 115–133.

[15] S. Haykin, Redes Neurais: Princípios e Práticas, Ed. Bookman, 2001.

[16] A. Solazzo, E. D. Sozzo, I. De Rose, M. D. Silvestri, G. C. Durelli and M. D. Santambrogio. Hardware design automation of convolutional neural networks. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 224-229.

[17] D. H. Hubel, T. N. Wiesel, Receptive fields of single neurones in the cat's striate cortex, *The Journal of Physiology* 148 (3), 1959, pp. 574–591.

[18] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, vol. 36, 1980, pp. 193-202.

[19] D. E Rumelhart, G.E. Hinton & R. J. Williams, Learning internal representations by backpropagating errors. In *Nature*, vol. 323, 1986, pp. 533–536.

[20] Y. LeCun, Y. Bengio, Word-level training of a handwritten word recognizer based on convolutional neural networks, in: *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, Vol. 2, IEEE, 1994, pp. 88–92.

[21] Y. Lecun, Y. Bengio, Convolutional networks for images, speech, and time-series, MIT Press, 1995.
[22] A. Krizhevsky, I. Sutskever, G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012, pp. 1097-1105.
[23] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, *Nature* 521 (7553), 2015, pp. 436-444.
[24] J. Gu, et al. Recent advances in convolutional neural networks. arXiv preprint arXiv:1512.07108, 2015.
[25] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting., *Journal of Machine Learning Research* 15 (1), 2014, pp. 1929–1958.
[26] M. Abadi, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software disponível em tensorflow.org.
[27] François Chollet. Comma.ai, 2016. URL http://keras.io/
[28] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. http://caffe.berkeleyvision.org/.
[29] R. Collobert, S. Bengio, and J. Mariethoz. Torch: a modular machine learning software library. Technical Report IDIAPRR 02-46, IDIAP, 2002.
[30] Y. Tang. Deep learning using support vector machines. CoRR, abs/1306.0239, v. 2, 2013.
[31] R. G. Congalton, A review of assessing the accuracy of classifications of remotely sensed data. *Remote Sensing of Environment*, vol. 37, 1991, pp. 35-46.
[32] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks. In *IEEE International Joint Conference on Neural Networks*, vol. 2, 2004, pp. 985–990.
[33] G. B. Huang, Q. Y. Zhu, and C. K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, vol. 70, 2006, pp. 489–501.