UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

DANIEL ARRUDA PIZZI

Análise da influência de descritores de textura LMP na entrada de redes neurais convolucionais

São Carlos
2017

DANIEL ARRUDA PIZZI

Análise da influência de descritores de textura LMP na entrada de redes neurais convolucionais

Monografia apresentada ao Curso de Engenharia Elétrica com ênfase em Eletrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista/Eletrônico.

Orientador: Prof. Dr. Adilson Gonzaga

São Carlos

2017

Pizzi, Daniel Arruda
P695a Análise da influência de descritores de textura LMP na entrada de redes neurais convolucionais / Daniel Arruda Pizzi; orientador Adilson Gonzaga. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Redes Neurais Convolucionais. 2. Descritores de textura. 3. Classificação de imagens. I. Título.

FOLHA DE APROVAÇÃO

Nome: Daniel Arruda Pizzi

Título: "Análise da influência de descritores de textura LMP na entrada de redes neurais convolucionais"

Trabalho de Conclusão de Curso defendido e aprovado em 23 / 11 / 2017,

com NOTA 7,7 (SETE, SETE), pela Comissão Julgadora:

Prof. Associado Adilson Gonzaga - Orientador - SEL/EESC/USP

Mestre Tamiris Trevisan Negri - Doutoranda - SEL/EESC/USP

Mestre Adriane Cavichiolli - Doutoranda - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

Este trabalho é dedicado aos meus pais, que nunca deixaram de me apoiar.

AGRADECIMENTOS

Agradeço ao meu orientador Prof. Dr. Adilson Gonzaga seu conhecimento compartilhado, assim como suas observações e direção que tornaram a execução desse trabalho possível.

Agradeço à minha família a estrutura que me foi proporcionada durante toda a minha vida.

Agradeço à minha namorada, Letícia, todo o amor, paciência e companheirismo durante essa minha jornada.

Agradeço aos colegas do LAVI o apoio durante essa trajetória.

Agradeço à Universidade de São Paulo.

RESUMO

PIZZI, D. A. **Análise da influência de descritores de textura LMP na entrada de redes neurais convolucionais**. 2017. 69p. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

O problema de classificação de imagens, tarefa na qual rotula-se uma imagem de entrada dentre um conjunto de categorias preestabelecidas, é recorrente e um dos mais importantes no campo de visão computacional. Nos últimos anos, os melhores resultados para esse problema têm sido obtidos com o uso de *redes neurais convolucionais* (*convolutional neural networks* - CNNs). Neste trabalho, avaliou-se o efeito do uso de descritores LMP (*Local Mapped Pattern*) aplicados na imagem de entrada. Primeiramente projetou-se uma arquitetura CNN que foi treinada utilizando-se o conjunto de imagens CIFAR-10, composto de 60.000 imagens de tamanho 32x32 pixels e 3 canais (RGB), sendo 50.000 imagens de treino e 10.000 imagens de teste divididas em 10 categorias. Após aferida a acurácia da arquitetura, verificou-se a variação dessa acurácia quando as imagens de entrada, tanto de treino quanto de teste, eram submetidas aos descritores LMP. Foram utilizados dois descritores LMP, o LMP sigmoidal e o LMP triangular, e foram testadas diferentes combinações entre a matriz de texturas obtidas por cada descritor e a imagem original. Após os testes, constatou-se que no caso da matriz de texturas ser calculada sobre cada canal a acurácia se manteve praticamente inalterada, mas quando a matriz de texturas era calculada sobre a matriz de intensidades da imagem há uma pequena melhora na acurácia da arquitetura somente devido a esse pré-processamento, já que a arquitetura se manteve inalterada. Em um campo de estudo relativamente novo como o de CNNs, em que ainda não se tem o controle total sobre quais parâmetros podem aumentar a eficiência da sua arquitetura, o uso de descritores LMP nas imagens de entrada de uma CNN se mostra como uma boa opção, ainda mais pelo fato de que não se necessita fazer alterações na arquitetura em si.

Palavras-chave: Redes neurais convolucionais. Descritores de textura. Classificação de imagens.

ABSTRACT

PIZZI, D. A. **Analysis of the influence of LMP texture descriptors in the input of convolutional neural networks.** 2017. 69p. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

The image classification problem, task in which is assigned to an input image a label from a pre-established set of categories, is recurrent and one of the core problems in computer vision. In the latest years, the best results for this problem have come from the use of convolutional neural networks (CNNs). In this study, the effect of using LMP (Local Mapped Pattern) texture descriptors in the input image was evaluated. In the first place a CNN architecture was designed and trained using the images dataset CIFAR-10, which consists of 60,000 images of size 32x32 pixels and 3 channels (RGB), being 50,000 training images and 10,000 test images, divided in 10 categories. After the architecture accuracy was measured, the variation of the accuracy was verified in the cases in which the input images (training and test images as well) were put under the influence of the LMP descriptors. Two LMP descriptors were utilized, the sigmoidal LMP and the triangular LMP, and also different combinations of the textures matrix obtained from the descriptors and the original input image. After the tests, it was found that in the case of the textures matrix to be calculated from each different channel of the input image, the architecture accuracy remained practically unaltered, but, in the case of the textures matrix to be calculated from the intensity matrix of the input image, there is a little improvement in the architecture accuracy from this pre-processing alone, once the architecture remained unaltered. In a relatively new field of study such as CNNs, in which it is not completely known what can improve the efficiency of a certain architecture, the use of LMP in the input of a CNN reveals itself as a good option for this task, even more so due to the fact that it does not require any changes in the architecture itself.

Key-words: Convolutional neural networks. Texture descriptors. Image classification.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LMP	–	Local Mapped Pattern
LMPs	–	Local Mapped Pattern-sigmoid
LMPt	–	Local Mapped Pattern-triangular
LAVI	–	Laboratório de Visão Computacional
LFP	–	Local Fuzzy Pattern
CNN	–	Convolutional Neural Network
ReLU	–	Rectified Linear Unit

SUMÁRIO

CAPÍTULO 1

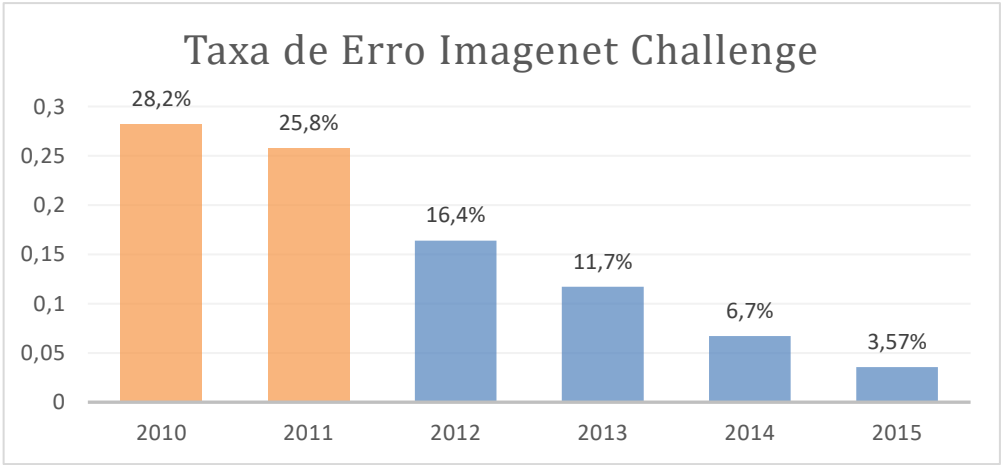
INTRODUÇÃO

Em visão computacional, a habilidade de se classificar uma imagem possui muitas utilidades no dia de hoje. Desde a classificação pura de imagens utilizada em websites de busca de imagens até o reconhecimento facial utilizado em redes sociais e como ferramenta de segurança para o desbloqueio de celulares, este campo de estudo tem encontrado cada vez mais usos práticos dessa tecnologia.

Um sistema de classificação de imagens é capaz de receber uma imagem de entrada e classificar essa imagem de acordo com categorias preestabelecidas [1]. A saída do sistema é um vetor em que cada componente representa a pontuação dessa imagem dentro de cada categoria. A categoria com a maior pontuação é escolhida pelo sistema como a categoria a que a imagem pertence.

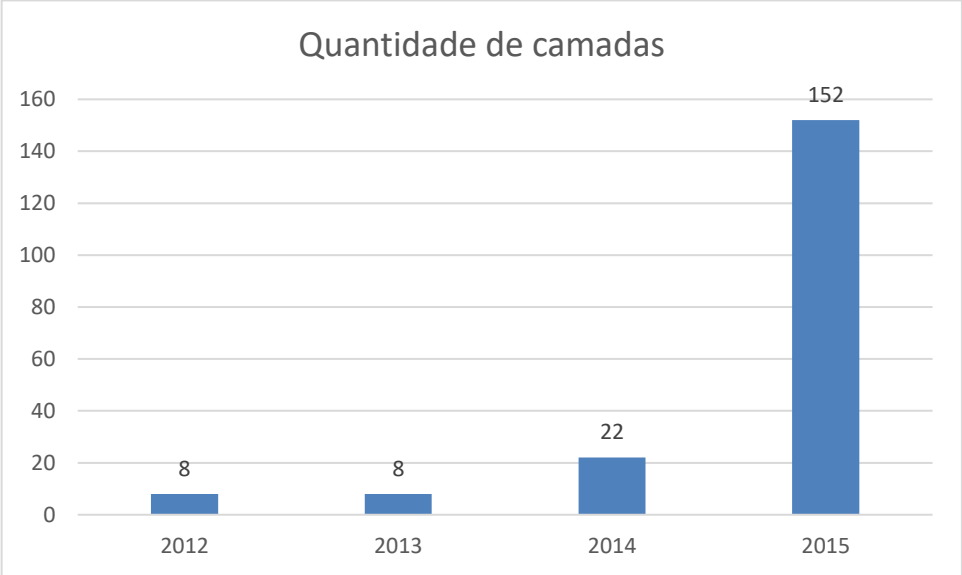
Nos últimos anos, o foco desse campo de estudo tem sido as redes neurais convolucionais (CNNs), principalmente desde 2012, quando a arquitetura AlexNet [2] ganhou o ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*), uma competição anual em que competem os melhores sistemas de classificação de imagens utilizando-se o conjunto de imagens ImageNet. A arquitetura de CNN AlexNet conseguiu uma melhora significante em relação ao vencedor de 2011 e desde a edição de 2012 todos os vencedores utilizam CNNs. A *Figura 1* [3] apresenta as texas de erro dos sistemas vencedores do ILSVRC de 2010 a 2015. Em azul todos os sistemas baseados em CNNs.

Figura 1: Taxa de erro dos vencedores ILSVRC de 2012 a 2015



Ano após ano, as taxas de erro dos sistemas baseados em CNNs têm caído vertiginosamente, com valores de 16,4% para a AlexNet [2] em 2012, 6,7% para a VGG [4] em 2014 e 3,5% para a ResNet [5] em 2015. Ocorre que, as arquiteturas das CNNs têm aumentado ano após ano, ficando cada vez mais profundas. Na *Figura 2*, pode-se observar a quantidade de camadas presente em cada arquitetura vencedora do ILSVRC de 2012 a 2015 [3].

Figura 2: Quantidade de camadas das arquiteturas vencedoras do ILSVRC de 2012 a 2015



1.1 Motivação

Dado que, para que uma rede neural convolucional consiga fazer a classificação de imagens ela deve passar por um treinamento prévio com o maior número de imagens possível, quanto mais camadas uma arquitetura possui, maior será o gasto computacional e, consequentemente, o tempo necessário para que se faça esse treinamento. Assim, este estudo procura formas de aumentar a acurácia dos sistemas de classificação de imagem sem mexer em sua arquitetura.

Considerando a possibilidade de se extrair informações das imagens de entrada e utilizar-se dessas novas informações, juntamente a essas imagens, para aumentar a acurácia do sistema e dado que o grupo de pesquisa do Laboratório de Visão Computacional (LAVI) da USP têm obtido resultados consistentes na classificação de texturas com a utilização do *Local Mapped Pattern* (LMP) [6][7][8], um descritor de texturas desenvolvido no laboratório, decidiu-se pela utilização do LMP para o cálculo de matrizes de texturas a partir das imagens de entrada a

serem utilizadas no sistema e sua utilização em conjunto com as mesmas como novas entradas para o sistema.

1.2 Objetivos

O objetivo deste trabalho é investigar os efeitos resultantes na acurácia de uma CNN pela utilização do descritor de texturas LMP para a extração e subsequente uso das informações de textura das imagens de entrada dessa CNN.

Para isso é criada uma arquitetura CNN que servirá de base para os testes. Primeiramente o treinamento e os testes são feitos com a imagens de entrada puras, sem nenhuma informação extra adicionada. Em seguida cria-se matrizes de texturas das imagens de entrada (tanto as de treinamento quanto as de teste) utilizando-se os descritores LMP. As informações destas matrizes são adicionadas às imagens originais e então o treinamento e os teste são repetidos para diversas combinações das matrizes de texturas com as imagens originais, para que se possa comparar os efeitos da utilização dos descritores de textura na acurácia do sistema.

1.3 Organização do trabalho

Os capítulos deste trabalho estão divididos conforme segue.

- No capítulo 2 é abordada brevemente a teoria envolvendo redes neurais convolucionais e sua arquitetura.
- No capítulo 3 é feita uma introdução ao conceito por trás dos descritores de textura LMP.
- No capítulo 4 é explanada a metodologia utilizada neste estudo.
- No capítulo 5 é feita uma discussão dos resultados obtidos.
- No capítulo 6 são feitas considerações finais e sugestões para trabalhos futuros.

CAPÍTULO 2

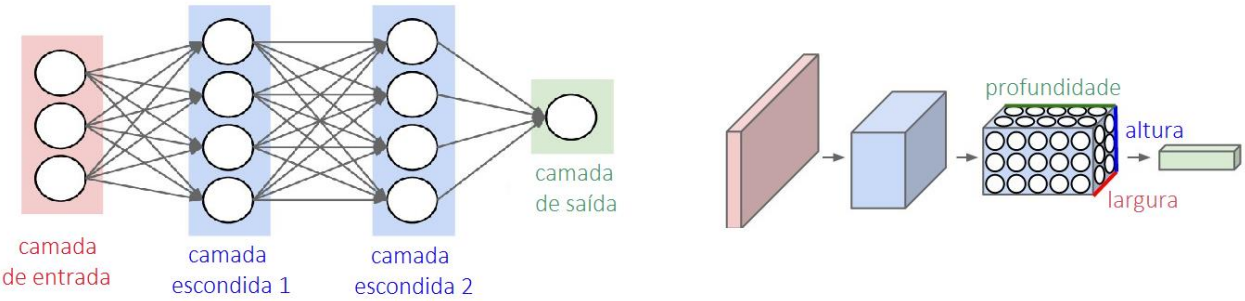
REDES NEURAIS CONVOLUCIONAIS

Uma rede neural comum possui como entrada um vetor. Para se trabalhar com imagens em uma rede neural comum, faz-se necessário primeiramente transformar a matriz que representa a imagem em um vetor. Isso significaria transformar uma imagem de tamanho 32x32 e 3 canais, por exemplo, em um vetor com 3.072 elementos. Se as imagens forem maiores então, esses vetores começam a assumir valores absurdos. Isso faz com que uma rede neural comum não seja a melhor alternativa para se trabalhar com imagens [1].

LeCun et al. (1999), faz um dos primeiros usos documentados de uma rede neural convolucional para tratamento de imagens. O trabalho utiliza CNNs com sucesso para reconhecer dígitos escritos à mão. O termo CNN vêm do inglês *Convolutional Neural Network*. Uma rede convolucional recebe como entrada a matriz que representa a imagem, e cada camada de convolução possuiu 3 dimensões, altura, largura e profundidade, o que torna CNN ótima para o uso em imagens.

A diferença entre os dois tipos de rede neural pode ser melhor visualizada na *Figura 3*.

Figura 3: Comparação rede neural comum e convolucional



Fonte: Notas de aula CS231n, Stanford University, 2017

2.1 Arquitetura

2.1.1 Entrada

A camada de entrada de uma rede convolucional é uma matriz que representa a imagem. A imagem deve ser quadrada, com a profundidade normalmente igual a 1, no caso de imagens em preto e branco, e 3, no caso de imagens coloridas.

2.1.2 Camada de Convolução

Uma camada de convolução vai ser formada por *n* filtros quadrados de tamanho $m\ x\ m$, com *m* e *n* sendo escolha do projetista da arquitetura e profundidade igual ao número de canais da entrada.

Um filtro de $5\ x\ 5\ x\ 3$ por exemplo, consegue "enxergar" um pedaço de $5\ x\ 5\ x\ 3$ na matriz da camada anterior. Este filtro é então "deslizado" por toda a matriz, de acordo com o passo desejado.

A saída de uma camada de convolução possui tamanho *[a x a x c x n]*, onde *c* é o número de canais da imagem, *n* é o número de filtros desta camada e *a* é definido pela Equação 1[1].

$$a = \frac{input - tam + 2 * pad}{pas} + 1, \qquad (1)$$

Onde *input* é o tamanho da entrada, *tam* é o tamanho do filtro de convolução, *pad* é a quantidade de padding de zeros (linhas e colunas adicionadas à matriz e preenchidas com zeros para se manter o tamanho da matriz na saída da aoperação) e *pas* é o tamanho do passo utilizado.

2.1.3 Função de Ativação

A função de ativação é uma função utilizada para se modelar a saída de um neurônio em redes neurais. Essa função determina se cada camada vai ter valor zero ou diferente de zero, a mesma forma que um neurônio transmite ou não pulsos elétricos. Hinton, Krizhevskye Sutskever (2012), indicam um caminho seguido por praticamente todas as CNNs em relação à função responsável por modelar a saída de um neurônio. Chamada primeiramente por Hinton e Nair (2010) de ReLU (Rectified Linear Units) a função de ativção utilizada segue a Fórmula 2.

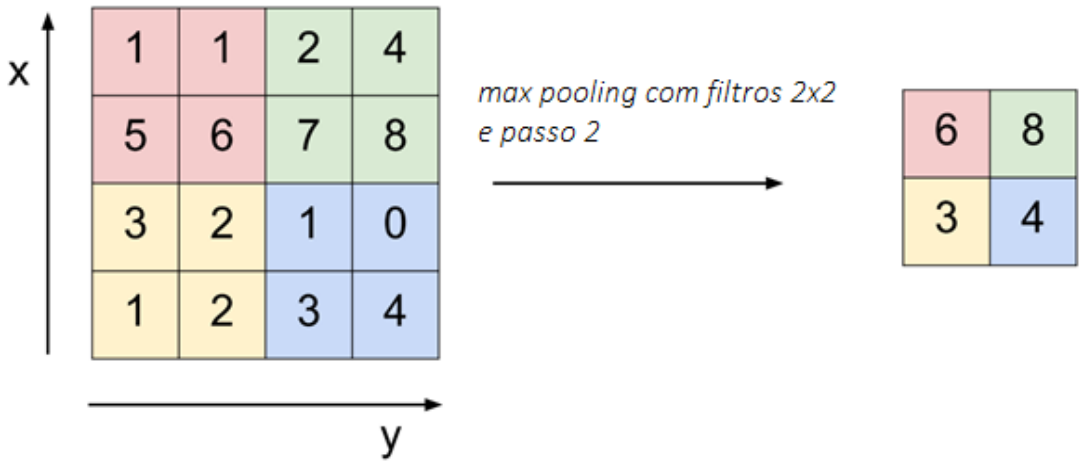
$$f(x) = \max(0, x) \qquad (2)$$

2.1.4 *Pooling*

Camadas de *pooling* são utilizadas para diminuir o tamanho das matrizes trabalhadas. No caso a profundidade não é afetada. A fórmula M também pode ser aplicada para calcular o tamanho da saída de uma camada de pooling.

Dois tipos de *pooling* são mais utilizados, *Max Pooling* calcula o valor máximo em uma vizinhança e *Average Pooling* calcula a média da vizinhança. *A Figura 4* torna fácil a visualização de como uma camada de *pooling* diminui o tamanho de sua entrada..

Figura 4: *Max Pooling*



Fonte: Notas de aula CS231n, Stanford University, 2017

2.1.5 Camada totalmente conectada

As camadas totalmente são formadas por um vetor de neurônios, como em uma rede neural comum. Elas são as útimas camadas da arquitetura e recebem todos os elementos da camada anterior. Sua saída é um vetor com o mesmo número de elementos que o vetor de neurônios que forma essa camada.

Se a camada anterior à camada completamente conectada for formada por 64 filtros de dimensão $[5\ x\ 5\ x\ 3]$, por exemplo, isso significa que a entrada dessa camada tem dimensão de $d = 64\ x\ 5\ x\ 5\ x\ 3 =$ 4.800 .

2.1.6 Classificador *Softmax*

O classificador *softmax* tem como entrada a saída de uma camada totalmente conectada. O classificador calcula uma função de perda (função que calcula quão distante do resultado esperado está o resultado obtido) para a pontuação que chega a ele de acordo com a fórmula:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

Em que f_{yi} refere-se à pontuação esperada para aquela classe e f_j à pontuação obtida [1].

2.2 Treinamento

O processo de treinamento de uma rede neural consiste em 2 processos, *forward propagation* e *backpropagation*. Durante a fase de *forward propagation* a imagem de entrada passa por todas as camadas, sofrendo as operações que cabem a cada uma delas. Ao fim de cada iteração, uma função de perda que relaciona os resultados obtidos com os resultados esperados é calculada.

2.2.1 *Backpropagation*

Após calculada a função de perda, inicia-se o processo de *backpropagation*, em que deriva-se a função de perda calculada em relação a cada matriz de filtros das camadas totalmente conectadas, derivada essa chamada de gradiente descendente, e das camadas de convolução e atualiza-se o valor dessas matrizes.

Algumas variáveis que influenciam diretamente o treinamento são definidas previamente. São elas:

- Taxa de aprendizado inicial: fator que determina a força da influência do gradiente descendente durante o processo de backpropagation.

- Fator de queda da taxa de aprendizado: A cada número de iterações passadas, diminui-se o valor da taxa de aprendizado. Isso acontece pois a cada iteração que passa, mais perto fica-se do resultado ótimo.

2.2 Regularização

Utiliza-se regularização em CNNs para evitar que o sistema fique moldado em excesso de acordo com os dados de treinamento e tenha dificuldades com imagens que não faziam parte do grupo de treinamento.

Neste trabalho utilizou-se a regularização L2, que adiciona um elemento quadrático das matrizes de pesos à função de perdas, penalizando matrizes com valores grandes. A regularização L2 segue a Equação 3:

$$R(W) = \sum_{k} \sum_{l} W_{k,l}^{2} \tag{3}$$

Em que *W,* representa todos os elementos das matrizes de convolução. Assim, a regularização L2 penaliza altos valores nessas matrizes. O elemento R(W) aparece na fórmula da função de perdas, multiplicada por um fator λ, que vai determinar a influência da regularização na função de perdas [1].

CAPÍTULO 3

DESCRITORES DE TEXTURA LMP (LOCAL MAPPED PATTERN)

Apresentado por VIEIRA et al. (2012) como um novo descritor de texturas baseado em lógicas *fuzzy* chamado LFP (*Local Fuzzy Pattern*), o descritor de texturas evoluiu para LMP (*Local Mapped Pattern*) (FERRAZ; PEREIRA; GONZAGA, 2014) para permitir o uso de qualquer função e não somente as funções de lógica *fuzzy*. VIEIRA et al. (2017) esclarece que o modelo "considera a soma das diferenças de cada nível de cinza de uma determinada vizinhança em torno do pixel central como um padrão local que pode ser mapeado para um histograma usando uma função de mapeamento".

Para este estudo, não se fará necessário mapear o histograma, somente a aplicação dos descritores LMP em cada imagem, para a obtenção do que será referido como uma matriz de texturas. Para isso, é calculada a soma das diferenças entre os níveis de cinza do pixel central e de sua vizinhança. Esse valor é inserido em uma função (diferente para o caso do LMP Sigmoidal e do LMP triangular). Ao se fazer isso em todos os pixels da imagem, tem-se uma matriz de texturas.

3.1 LMP Sigmoidal

Para análise de texturas utilizando-se o LMP sigmoidal, utiliza-se a Equação 4, função com o formato de uma onda sigmoide.

$$f_g(i,j) = \frac{1}{1+e^{\frac{-[A(k,l)-g(i,j)]}{\beta}}} \tag{4}$$

TANIWAKI et al. (2016) afirma que nesta fórmula, a componente β vai definir a inclinação da curva, A(k,l) será o nível de cinza da borda da vizinhança e g(i,j) será o nível de cinza do pixel central da vizinhança.

3.1 LMP Triangular

Neste caso o descritor LMP funcionará como um detector de bordas e utiliza-se a Equação 5.

$$f_{g(i,j)} = max\left(0{,}1 - \frac{|g(i,j) - A(k,l)|}{\delta}\right) \quad (5)$$

E δ representa o espalhamento de número fuzzy, assim, quão maior o valor de δ, maior a vizinhança levada em conta para cada cálculo.

CAPÍTULO 4

METODOLOGIA

O treinamento e os teste foram realizados utilizando-se a *toolbox* de redes neurais *Neural Network*, disponível no software MATLAB versão R2017a, que facilita a montagem de sua arquitetura projetada e seus testes. O MATLAB foi rodado em uma CPU Intel i7-6700, que conseguiu fazer os treinamentos em um período razoável, em torno de 20 minutos para cada treimanento..

4.1 Conjunto de imagens utilizado

Primeiramente, escolheu-se o conjunto de imagens a serem utilizadas para o treinamento e teste da arquitetura.

Escolheu-se o conjunto de imagens CIFAR-10 [3]. Este conjunto é formado por 60.000 imagens, sendo 50.000 imagens de treino e 10.000 imagens de teste. As imagens possuem tamanho 32x32 pixels e estão no formato RGB, ou seja, possuem 3 canais.

As imagens do conjunto CIFAR-10 são divididas em 10 categorias: aviões, carros, pássaros, gatos, cervos, cachorros, sapos, cavalos, navios e caminhões.

Este conjunto de imagens foi escolhido pelo tamanho reduzido das imagens (32x32 pixels), e pelo número reduzido de imagens (60.000), o que torna os treinamentos e testes mais rápidos.

4.2 Arquitetura e opções de treinamento

Foi projetada uma arquitetura simples que pudesse ser treinada em um tempo razoável. A arquitetura escolhida possui 17 camadas sendo 1 camada de entrada, 4 camadas de convolução, 3 camadas de *pooling*, 6 camadas de ativação e 3 camadas completamente conectadas. A saída da arquitetura passa por uma função *softmax* para a classificação dos resultados e uma camada de classificação escolhe o melhor resultado e assume a categoria para a imagem.

A seguir são descritos cada camada da CNN.

Layer 1: Input - Camada de entrada.

Layer 2: Conv 1 – Camada de convolução formada por 32 filtros de tamanho 5x5x3, com passo 1 e padding 2.

Layer 3: Max Pooling 1 – Camada de max pooling 3x3 com passo 2 e padding 0.

Layer 4: ReLu – Camada de ativação.

Layer 5: Conv 2 - Camada de convolução formada por 64 filtros de tamanho 5x5x3, com passo 1 e padding 2.

Layer 6: ReLu – Camada de ativação.

Layer 7: Max Pooling 2 – Camada de max pooling 3x3 com passo 2 e padding 0.

Layer 8: Conv 3 - Camada de convolução formada por 128 filtros de tamanho 3x3x3, com passo 1 e padding 1.

Layer 9: ReLu – Camada de ativação.

Layer 10: Average Pooling 1 – Camada de average pooling 3x3 com passo 2 e padding 0.

Layer 11: Conv 4 - Camada de convolução formada por 256 filtros de tamanho 3x3x3, com passo 1 e padding 1.

Layer 12: ReLu – Camada de ativação.

Layer 13: FC1 (*fully connected)* – Camada totalmente conectada 1 com 576 nêuronios.

Layer 14: ReLu – Camada de ativação.

Layer 15: FC2 – Camada totalmente conectada 2 com 64 nêuronios.

Layer 16: ReLu – Camada de ativação.

Layer 17: FC3 – Camada totalmente conectada 3 com 10 nêuronios.

Os valores de hiperparâmetros utilizados foram os mesmos hiperparâmetros utilizados no exemplo Demo_trainingfromscratch[9].

Taxa de aprendizado inicial = 0,001;

Fator de queda da taxa de aprendizado = 0,1;

$\lambda = 0{,}004$.

4.3 Treinamentos e Testes

4.3.1 Treinamento e Teste 1

O primeiro treinamento e teste foram feitos utilizando o conjunto de imagens CIFAR-10 sem nenhuma mudança às imagens. O teste é feito com as 10.000 imagens do CIFAR-10 e nesse passo é calculada a acurácia (número de acertos sobre o número total de testes) da CNN.

4.3.2 Treinamento e Teste 2

Para o segundo treinamento e subsequente teste as imagens de entrada foram modificadas utilizando o descritor de texturas LMPs. Após a realização de 3 testes para verificar o melhor resultado obtido, tanto para o valor de β quanto o valor de δ foram utilizados o valor de 1.

Primeiramente, aplicou-se um padding com zeros de tamanho 2 em todas as imagens para que elas mativessem o tamanho 32x32x3 após passar pelo descritor LMP. Ao aplicar o padding cada imagem ficou com tamnho 34x34x3.

Em seguida aplicou-se o descritor LMPs em cada canal dos 3 canais de cada imagem, em todas as 60.000 imagens do conjunto. Assim cada, imagem passou a possuir uma matriz de texturas de tamanho $32x32x3$. Para esse passo, utilizou-se o seguinte algoritmo [4]:

```
for i = 1:50000
for k = 1:3
input = padarray(dados_tr(:,:,k,i),1,'replicate');
 input = padarray(input',1,'replicate');
 input = input';
 matriz_treino_sig(:,:,k,i) = LMP(input,3, 1);
end
end
```

Após ter-se calculado a matriz de texturas da imagem, calculou-se a média aritmética, elemento a elemento das imagens originais com suas matrizes de texturas. Para esse passo, utilizou-se o seguinte algoritmo. Essas são as novas imagens de entrada:

```
for z = 1:50000
for k = 1:3
 for j = 1:32
   for i = 1:32
dados_tr(i,j,k,z) = (dados_tr(i,j,k,z) + matriz_treino_sig (i,j,k,z))/2;
```

```
            end
        end
    end
end
```

Em seguida repetiram-se o treinamento e os testes, dessa vez com as novas imagens.

4.3.3 Treinamento e Teste 3

Esse passo repete o Treinamento e Teste 2 , com a única diferença de se utilizar o LMP triangular em vez do sigmoidal.

4.3.4 Treinamento e Teste 4

Para o quarto treinamento e subsequente teste as imagens de entrada foram modificadas utilizando-se mais uma vez o descritor de texturas LMPs.

Primeiramente, calculou-se a matriz de intensidades de cada imagem, utilizando-se da função rgb2gray disponibilizada pelo Matlab. Essa função segue o algoritmo:

$$0.2989 * R + 0.5870 * G + 0.1140 * B \quad [10]$$

Em que R, G e B são os valores de cada canal em cada pixel da imagem.

Em seguida, aplicou-se um padding com zeros de tamanho 2 em todas as imagens para que elas mativessem o tamanho 32x32x3 após passar pelo descritor LMP. Ao aplicar o padding cada imagem ficou com tamnho 34x34x3.

Em seguida, aplicou-se o descritor LMPs na matriz de intensidades calculada em todas as 60.000 imagens do conjunto. Assim cada imagem possui agora uma matriz de texturas de tamanho 32x32x1. Para esse passo, utilizou-se o seguinte algoritmo:

```
for i = 1:50000
    input = padarray(treino_bw(:,:,i),1,'replicate');
    input = padarray(input',1,'replicate');
    input = input';
```

```
    matriz_treino_sig(:,:,i) = LMP(input,3, 1);
end
```

Após ter-se calculado a matriz de texturas da imagem, calculou-se a média aritmética, elemento a elemento das imagens originais com suas matrizes de texturas. Para esse passo, utilizou-se o mesmo algoritmo utilizado para o treinamento 2.

Em seguida repetiram-se o treinamento e os testes, dessa vez com as novas imagens.

4.3.5 Treinamento e Teste 5

Esse passo repete o Treinamento e Teste 4, com a única diferença de se utilizar o LMP triangular em vez do sigmoidal.

4.3.6 Treinamento e Teste 6

Para os últimos treinamento e teste as imagens a serem utilizadas na entrada foram calculadas a partir da média entre as imagens originais e a média entre as matrizes de textura LMPs e LMPt.

A Tabela 1 apresenta uma comparação entre cada treinamento e teste realizado.

Treinamento e teste	LMP	Aplicação
1	-	-
2	Sigmoidal	Por canal
3	Triangular	Por canal
4	Sigmoidal	Matriz de intensidades
5	Triangular	Matriz de intensidades
6	Sigmoidal e Triangular	Matriz de intensidades

Tabela 1: Comparação entre treinamentos e testes realizados

CAPÍTULO 5

RESULTADOS

Neste capítulo são apresentados os resultados dos treinamentos e testes realizados, com uma análise dos mesmos.

5.1 Imagens Resultantes dos pré-processamentos.

Na Figura 5 pode-se ver todas as imagens resultantes do uso dos descritores de textura LMP sobre uma imagem do conjunto CIFAR-10.

Figura 5: Imagens resultantes dos pré-processamentos

Legenda da Figura 5:

1. Imagem original;
2. Matriz de intensidades;
3. Matriz de texturas calculada utilizando o descritor de textura LMPs aplicado nos 3 canais;
4. Média entre a matriz de texturas LMPs aplicada nos 3 canais e a imagem original;
5. Matriz de texturas calculada utilizando o descritor de textura LMPt aplicado nos 3 canais;
6. Média entre a matriz de texturas LMPt aplicada nos 3 canais e a imagem original;
7. Matriz de texturas calculada utilizando o descritor de textura LMPs aplicado na matriz de intensidades;
8. Média entre a matriz de texturas LMPs aplicada na matriz de intensidades e a imagem original;
9. Matriz de texturas calculada utilizando o descritor de textura LMPt aplicado na matriz de intensidades;

10. Média entre a matriz de texturas LMPt aplicada na matriz de intensidades e a imagem original;
11. Média entre as matrizes de texturas LMPs e LMPt aplicados na matriz de intensidades;
12. Média entre a matriz resultante da média entre as matrizes de texturas LMPs e LMPt e a imagem original.

5.2 Acurácia obtida em cada treinamento

Na Tabela 2 pode-se ver a acurácia resultante de cada treinamento e a variação sendo de acurácia em relação ao Teste 1 (que utiliza como entrada da arquitetura as imagens originais):

Teste	Acurácia	Variação da acurácia em relação ao Teste 1
Teste 1 (Imagem Original)	77,72%	-
Teste 2 (LMPs, 3 canais)	77,71%	-0,01%
Teste 3 (LMPt, 3 canais	77,95%	0,23%
Teste 4 (LMPs, matriz de intensidades)	78,17%	0,45%
Teste 5 (LMPt, matriz de intensidade)	78,39%	0,67%
Teste 6 (LMPs + LMPt, matriz de intensidades)	78,45%	0,73%

Tabela 1: Precisão e variação em relação ao Teste 1

5.3 Análise dos resultados obtidos

O Teste 1 alcançou uma acurácia de 77,72% , o que é um resultado bastante justo para essa arquitetura simples. De qualquer forma, esta arquitetura foi projetada apenas para servir de base para que se pudesse observar a influência dos descritores de textura.

Os Testes 2 e 3, referentes à aplicação dos descritores LMPs e LMPt sobre cada canal RGB da imagem não foram positivos, obtendo praticamente a mesma acurácia para o caso do LMPs e variação positiva na acurácia de 0,23% para o caso do LMPt.

Os Testes 4 e 5, referentes à aplicação dos descritores LMPs e LMPt sobre a matriz de intensidades foram um pouco melhores, obtendo uma variação positiva de 0,45% para o caso LMPs e de 0,67% para o caso LMPt.

O Teste 6, referente ao treinamento que combinou as matrizes de textura que obtiveram os 2 melhores resultados conseguiu a melhor acurácia de todas, 78,45% , uma variação positiva de acurácia de 0,73% em relação ao Teste 1.

Não é possível verificar os motivos pelos quais os resultados de alguns treinamentos são melhores que outros, mas apesar das pequenas melhoras obtidas, percebe-se que os resultados seguiram alguns padrões, o que indica que esses resultados não são aleatórios. O primeiro padrão percebido foi que aplicação dos descritores de textura na matriz de intensidades obteve melhores resultados que a aplicação em cada canal separadamente. O segundo padrão percebido foi que a utilização do descritor LMPt, que realça as altas frequências da imagem obteve resultados um pouco melhores do que o descritor LMPs. Finalmente, o uso de uma combinação das matrizes de textura que obtiveram os melhores resultados resultou no melhor resultado do teste, o que indica um bom caminho a ser seguido.

CAPÍTULO 6

CONCLUSÃO

6.1 Considerações finais

Este trabalho se propôs avaliar a influência de descritores de textura LMP quando estes são utilizados para modificar as entradas de arquiteturas de CNNs com o objetivo de classificar imagens. Por ser um campo de estudo relativamente novo, não se sabe exatamente de que forma cada mudança feita no sistema como um todo, seja ela na arquitetura em si ou nas imagens de entrada, irá influenciar no resultado obtido. Todo tipo de teste que note algum padrão ou tendência de comportamento da rede neural é bem vindo. Dessa forma, este estudo foi bem sucedido pois encontrou alguns padrões de comportamento das redes neurais quando da utilização dos descritores de textura na entrada.

As melhoras obtidas na acurácia do sistema podem ser consideradas pequenas, chegando até a casa dos 0,7% de melhora, porém os testes foram feitos em uma arquitetura bem mais simples do que as arquiteturas profundas em uso atualmente. Se as mesmas mudanças na entrada forem aplicadas e utilizadas para treinar e testar uma rede profunda talvez essas melhoras possam ser mais significativas.

Um ponto interessante foi que o resultado obtido com a aplicação dos descritores em um só canal, na matriz de intensidades foi consistentemente melhor que o resultado obtido aplicando-se os descritores em cada canal separadamente. Esse foi um resultado diferente do esperado, pois imaginava-se que aplicando os descritores em cada canal, mais informações da imagem estavam sendo utilizadas que isso se refletiria em um resultado melhor, o que reforça que o comportamento das CNNs não é intuitivo.

6.2 Sugestões para trabalhos futuros

Devido ao elevado tempo de treinamento de cada teste, este trabalho não verificou os resultados utilizando-se diversos valores para os parâmetros β para o LMPs e δ para o LMPt. Uma ideia seria adicionar ao treinamento da rede uma camada que o backpropagation também atualizasse os valores de β e δ, atualizando a cada iteração as matrizes de texturas, otimizando esses parâmetros e o resultado final.

REFERÊNCIAS

[1] LI, F.; JOHNSON, J.;YEUNG,S.; **Convolutional Neural Networks for Visual Recognition**, Stanford University, 2017. Notas de Aula. Slides.

[2] HINTON,G.E.;KRIZHEVSKY, A,; SUTSKEVER,I. ImageNet Classification with Deep Convolutional Neural Networks. 2012. 9p. **Advances in neural information processing systems,** 1097-1105.

[3] RUSSAKOVSKY, O. et al. ImageNet Large Scale Visual Recognition Challenge. IJCV.2015. 42p. **International Journal of Computer Vision. Issue 3,** 211-252.
.
[4] SIMONYAN, K.; ZISSERMAN, A. Very Deep Convolutional Networks for Large-Scale Image Recognition, **arXiv preprint arXiv**:1409.1556, 2014.

[5] HE, K. et al. Deep Residual Learning for Image Recognition, 2016, 9p. **The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, 770-778.

[6] VIEIRA, R. T. **Análise de micropadrões em imagens digitais baseada em números fuzzy.** 2013. 151f. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2013.

[7] VIEIRA, R.T. **Descritores robustos à rotação de texturas baseados na abordagem LMP com acréscimo da informação de Magnitude e Sinal**. 2017. 156p. Tese (Doutorado em Ciências) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

[8] FERRAZ, C. T. **Novos descritores de textura para a localizaçãoe identificação de objetos em imagens usando "Bag of features".** 151p. Tese de doutorado – Escola de Engenharia de São Carlos, Universidade de São Paulo, 2016.

[9] HA, G. Deep Learning with MATLAB: Training a Neural Network from Scratch with MATLAB. Disponível em: https://www.mathworks.com/videos/training-a-neural-network-from-scratch-with-matlab-1492008542195.html. Acesso em 25 de out. 2017.

[10] HINTON,G.E.;NAIR, V. Rectified Linear Units Improve Restricted Boltzmann, 2010. 8p. **Proceedings, 27th International Conference on Machine Learning (ICML-10) .**807-814.

[6] TANIWAKI, R. **Aplicção do descritor LMP para o reconhecimento facial.** 2016. 61f. Monografia – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2013.

Apêndice A – Rotina para determinar, treinar e testar arquitetura

```
% Escolha da pasta que contém as imagens de treinamento
categories = {'Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog',
'Frog','Horse','Ship','Truck'};
rootFolder = 'cifar10Train';
imds = imageDatastore(fullfile(rootFolder, categories), ...
    'LabelSource', 'foldernames');


% Definição dos layers da arquitetura
    varSize = 32;
conv1 = convolution2dLayer(5,varSize,'Padding',2,'BiasLearnRateFactor',2);
conv1.Weights = gpuArray(single(randn([5 5 3 varSize])*0.0001));
fc1 = fullyConnectedLayer(576,'BiasLearnRateFactor',2);
fc1.Weights = gpuArray(single(randn([576 2304])*0.1));
fc2 = fullyConnectedLayer(64,'BiasLearnRateFactor',2);
fc2.Weights = gpuArray(single(randn([64 576])*0.1));
fc3 = fullyConnectedLayer(10,'BiasLearnRateFactor',2);
fc3.Weights = gpuArray(single(randn([10 64])*0.1));
layers = [
    imageInputLayer([varSize varSize 3]);
    conv1;
    maxPooling2dLayer(3,'Stride',2);
    reluLayer();
    convolution2dLayer(5,64,'Padding',2,'BiasLearnRateFactor',2);

    reluLayer();
    maxPooling2dLayer(3,'Stride',2);
    convolution2dLayer(3,128,'Padding',1,'BiasLearnRateFactor',2);
    reluLayer();
    averagePooling2dLayer(3,'Stride',2);
    convolution2dLayer(3,256,'Padding',1,'BiasLearnRateFactor',2);
    reluLayer();
    fc1;
    reluLayer();
    fc2;
    reluLayer();
    fc3;
    softmaxLayer()
    classificationLayer()];


% Definição das opções de treinamento
        opts = trainingOptions('sgdm', ...
    'InitialLearnRate', 0.001, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.1, ...
    'LearnRateDropPeriod', 8, ...
    'L2Regularization', 0.004, ...
    'MaxEpochs', 10, ...
```

```
    'MiniBatchSize', 100, ...
    'Verbose', true);


% Treinamento
    [net, info] = trainNetwork(imds, layers, opts);

% Carrega imagens teste
rootFolder = 'cifar10Test';
imds_test = imageDatastore(fullfile(rootFolder, categories), ...
    'LabelSource', 'foldernames');

% Testa todas as imagens e calcula acurácia
confMat = confusionmat(imds_test.Labels, labels);
confMat = confMat./sum(confMat,2);
mean(diag(confMat))
```

Apêndice B – Rotina para processar imagens aplicando LMP nos 3 canais

```
clear all;
close all;
clc;


% Importar dados e labels treinamento CIFAR-10

S1 = load('data_batch_1.mat');
S2 = load('data_batch_2.mat');
S3 = load('data_batch_3.mat');
S4 = load('data_batch_4.mat');
S5 = load('data_batch_5.mat');

% Montar matriz com dados

S1D = S1.data;
S2D = S2.data;
S3D = S3.data;
S4D = S4.data;
S5D = S5.data;

dados_tr0(1:10000,1:3072) = S1D;
dados_tr0(10001:20000,1:3072) = S2D;
dados_tr0(20001:30000,1:3072) = S3D;
dados_tr0(30001:40000,1:3072) = S4D;
dados_tr0(40001:50000,1:3072) = S5D;


for z = 1:50000
    for k = 1:3
        for i = 1:32
            for j = 1:32
                dados_tr(i,j,k,z) = dados_tr0(z,j+(i-1)*32+(k-1)*1024);
            end
        end
    end
end



% Aplicando LMP

for i = 1:50000
for k = 1:3
    input = padarray(dados_tr(:,:,k,i),1,'replicate');
    input = padarray(input',1,'replicate');
    input = input';
    matriz_treino_sig(:,:,k,i) = LMP(input,3, 1);
end
end

% Média dos canais com Matriz LMP

for z = 1:50000
    for k = 1:3
```

```
        for j = 1:32
            for i = 1:32
                dados_tr(i,j,k,z) = (dados_tr(i,j,k,z) + matriz_treino_sig
(i,j,k,z))/2;
            end
        end
    end
end

% Voltando ao formato inicial


for z = 1:50000
    for i = 1:32
            for j = 1:32
                 matriz_treino_sigmedia1(z,(i-1)*32+j) = dados_tr(i,j,1,z);
            end
    end
end

for z = 1:50000
    for i = 1:32
            for j = 1:32
                 matriz_treino_sigmedia2(z,(i-1)*32+j) = dados_tr(i,j,2,z);
            end
    end
end

for z = 1:50000
    for i = 1:32
            for j = 1:32
                 matriz_treino_sigmedia3(z,(i-1)*32+j) = dados_tr(i,j,3,z);
            end
    end
end

    MBD1(1:10000,1:1024) = matriz_treino_sigmedia1(1:10000,1:1024);
    MBD1(1:10000,1025:2048) = matriz_treino_sigmedia2(1:10000,1:1024);
    MBD1(1:10000,2049:3072) = matriz_treino_sigmedia3(1:10000,1:1024);

    MBD2(1:10000,1:1024) = matriz_treino_sigmedia1(10001:20000,1:1024);
    MBD2(1:10000,1025:2048) = matriz_treino_sigmedia2(10001:20000,1:1024);
    MBD2(1:10000,2049:3072) = matriz_treino_sigmedia3(10001:20000,1:1024);

    MBD3(1:10000,1:1024) = matriz_treino_sigmedia1(20001:30000,1:1024);
    MBD3(1:10000,1025:2048) = matriz_treino_sigmedia2(20001:30000,1:1024);
    MBD3(1:10000,2049:3072) = matriz_treino_sigmedia3(20001:30000,1:1024);

    MBD4(1:10000,1:1024) = matriz_treino_sigmedia1(30001:40000,1:1024);
    MBD4(1:10000,1025:2048) = matriz_treino_sigmedia2(30001:40000,1:1024);
    MBD4(1:10000,2049:3072) = matriz_treino_sigmedia3(30001:40000,1:1024);

    MBD5(1:10000,1:1024) = matriz_treino_sigmedia1(40001:50000,1:1024);
    MBD5(1:10000,1025:2048) = matriz_treino_sigmedia2(40001:50000,1:1024);
    MBD5(1:10000,2049:3072) = matriz_treino_sigmedia3(40001:50000,1:1024);
```

```
    data = MBD1;
    labels = load('data_batch_1.mat','labels');
    labels = labels.labels;
    save('new_data_batch_1','data','labels');

    data = MBD2;
    labels = load('data_batch_2.mat','labels');
    labels = labels.labels;
    save('new_data_batch_2','data','labels');

    data = MBD3;
    labels = load('data_batch_3.mat','labels');
    labels = labels.labels;
    save('new_data_batch_3','data','labels');

    data = MBD4;
    labels = load('data_batch_4.mat','labels');
    labels = labels.labels;
    save('new_data_batch_4','data','labels');

    data = MBD5;
    labels = load('data_batch_5.mat','labels');
    labels = labels.labels;
    save('new_data_batch_5','data','labels');

% Importar dados e labels TESTE CIFAR-10

T = load('test_batch.mat');

% Montar matriz com dados

TD = T.data;


dados_te0(1:10000,1:3072) = TD;



for z = 1:10000
    for k = 1:3
        for i = 1:32
            for j = 1:32
                dados_te(i,j,k,z) = dados_te0(z,j+(i-1)*32+(k-1)*1024);
            end
        end
    end
end




% Aplicando LMP
 for k = 1:3
for i = 1:10000
    input = padarray(dados_te(:,:,k,i),1,'replicate');
    input = padarray(input',1,'replicate');
    input = input';
    matriz_teste_sig(:,:,k,i) = LMP(input,3, 1);
```

```
end
end

% Média dos canais com Matriz LMP

for z = 1:10000
    for k = 1:3
        for j = 1:32
            for i = 1:32
                dados_te(i,j,k,z) = (dados_te(i,j,k,z) + matriz_teste_sig
(i,j,k,z))/2;
            end
        end
    end
end


% Voltando ao formato inicial


for z = 1:10000
    for i = 1:32
            for j = 1:32
                 matriz_teste_sigmedia1(z,(i-1)*32+j) = dados_te(i,j,1,z);
            end
    end
end

for z = 1:10000
    for i = 1:32
            for j = 1:32
                 matriz_teste_sigmedia2(z,(i-1)*32+j) = dados_te(i,j,2,z);
            end
    end
end

for z = 1:10000
    for i = 1:32
            for j = 1:32
                 matriz_teste_sigmedia3(z,(i-1)*32+j) = dados_te(i,j,3,z);
            end
    end
end

    MT1(1:10000,1:1024) = matriz_teste_sigmedia1(1:10000,1:1024);
    MT1(1:10000,1025:2048) = matriz_teste_sigmedia2(1:10000,1:1024);
    MT1(1:10000,2049:3072) = matriz_teste_sigmedia3(1:10000,1:1024);


    data = MT1;
    labels = load('test_batch.mat','labels');
    labels = labels.labels;
    save('new_test_batch','data','labels');
```

Apêndice C – Rotina para processar imagens aplicando LMP na matriz de intensidades

```
clear all;
close all;
clc;


% Importar dados e labels treinamento CIFAR-10

S1 = load('data_batch_1.mat');
S2 = load('data_batch_2.mat');
S3 = load('data_batch_3.mat');
S4 = load('data_batch_4.mat');
S5 = load('data_batch_5.mat');

% Montar matriz com dados

S1D = S1.data;
S2D = S2.data;
S3D = S3.data;
S4D = S4.data;
S5D = S5.data;

dados_tr0(1:10000,1:3072) = S1D;
dados_tr0(10001:20000,1:3072) = S2D;
dados_tr0(20001:30000,1:3072) = S3D;
dados_tr0(30001:40000,1:3072) = S4D;
dados_tr0(40001:50000,1:3072) = S5D;


for z = 1:50000
    for k = 1:3
        for i = 1:32
            for j = 1:32
                dados_tr(i,j,k,z) = dados_tr0(z,j+(i-1)*32+(k-1)*1024);
            end
        end
    end
end



% Aplicando LMP

for i = 1:50000
for k = 1:3
    input = padarray(dados_tr(:,:,k,i),1,'replicate');
    input = padarray(input',1,'replicate');
    input = input';
    matriz_treino_delt(:,:,k,i) = LMPt(input,3, 1);
end
end
```

```
% Média dos canais com Matriz LMP

for z = 1:50000
    for k = 1:3
        for j = 1:32
            for i = 1:32
                dados_tr(i,j,k,z) = (dados_tr(i,j,k,z) + matriz_treino_delt
(i,j,k,z))/2;
            end
        end
    end
end

% Voltando ao formato inicial


for z = 1:50000
    for i = 1:32
            for j = 1:32
                 matriz_treino_deltmedia1(z,(i-1)*32+j) =
dados_tr(i,j,1,z);
            end
    end
end

for z = 1:50000
    for i = 1:32
            for j = 1:32
                 matriz_treino_deltmedia2(z,(i-1)*32+j) =
dados_tr(i,j,2,z);
            end
    end
end

for z = 1:50000
    for i = 1:32
            for j = 1:32
                 matriz_treino_deltmedia3(z,(i-1)*32+j) =
dados_tr(i,j,3,z);
            end
    end
end

    MBD1(1:10000,1:1024) = matriz_treino_deltmedia1(1:10000,1:1024);
    MBD1(1:10000,1025:2048) = matriz_treino_deltmedia2(1:10000,1:1024);
    MBD1(1:10000,2049:3072) = matriz_treino_deltmedia3(1:10000,1:1024);

    MBD2(1:10000,1:1024) = matriz_treino_deltmedia1(10001:20000,1:1024);
    MBD2(1:10000,1025:2048) = matriz_treino_deltmedia2(10001:20000,1:1024);
    MBD2(1:10000,2049:3072) = matriz_treino_deltmedia3(10001:20000,1:1024);

    MBD3(1:10000,1:1024) = matriz_treino_deltmedia1(20001:30000,1:1024);
    MBD3(1:10000,1025:2048) = matriz_treino_deltmedia2(20001:30000,1:1024);
    MBD3(1:10000,2049:3072) = matriz_treino_deltmedia3(20001:30000,1:1024);

    MBD4(1:10000,1:1024) = matriz_treino_deltmedia1(30001:40000,1:1024);
    MBD4(1:10000,1025:2048) = matriz_treino_deltmedia2(30001:40000,1:1024);
    MBD4(1:10000,2049:3072) = matriz_treino_deltmedia3(30001:40000,1:1024);
```

```
    MBD5(1:10000,1:1024) = matriz_treino_deltmedia1(40001:50000,1:1024);
    MBD5(1:10000,1025:2048) = matriz_treino_deltmedia2(40001:50000,1:1024);
    MBD5(1:10000,2049:3072) = matriz_treino_deltmedia3(40001:50000,1:1024);



    data = MBD1;
    labels = load('data_batch_1.mat','labels');
    labels = labels.labels;
    save('new_data_batch_1','data','labels');

    data = MBD2;
    labels = load('data_batch_2.mat','labels');
    labels = labels.labels;
    save('new_data_batch_2','data','labels');

    data = MBD3;
    labels = load('data_batch_3.mat','labels');
    labels = labels.labels;
    save('new_data_batch_3','data','labels');

    data = MBD4;
    labels = load('data_batch_4.mat','labels');
    labels = labels.labels;
    save('new_data_batch_4','data','labels');

    data = MBD5;
    labels = load('data_batch_5.mat','labels');
    labels = labels.labels;
    save('new_data_batch_5','data','labels');

% Importar dados e labels TESTE CIFAR-10

T = load('test_batch.mat');

% Montar matriz com dados

TD = T.data;


dados_te0(1:10000,1:3072) = TD;



for z = 1:10000
    for k = 1:3
        for i = 1:32
            for j = 1:32
                dados_te(i,j,k,z) = dados_te0(z,j+(i-1)*32+(k-1)*1024);
            end
        end
    end
end
```

```
% Aplicando LMPt
 for k = 1:3
for i = 1:10000
    input = padarray(dados_te(:,:,k,i),1,'replicate');
    input = padarray(input',1,'replicate');
    input = input';
    matriz_teste_delt(:,:,k,i) = LMPt(input,3, 1);
end
end

% Média dos canais com Matriz LMPt

for z = 1:10000
    for k = 1:3
        for j = 1:32
            for i = 1:32
                dados_te(i,j,k,z) = (dados_te(i,j,k,z) + matriz_teste_delt
(i,j,k,z))/2;
            end
        end
    end
end


% Voltando ao formato inicial


for z = 1:10000
    for i = 1:32
            for j = 1:32
                 matriz_teste_deltmedia1(z,(i-1)*32+j) = dados_te(i,j,1,z);
            end
    end
end

for z = 1:10000
    for i = 1:32
            for j = 1:32
                 matriz_teste_deltmedia2(z,(i-1)*32+j) = dados_te(i,j,2,z);
            end
    end
end

for z = 1:10000
    for i = 1:32
            for j = 1:32
                 matriz_teste_deltmedia3(z,(i-1)*32+j) = dados_te(i,j,3,z);
            end
    end
end

    MT1(1:10000,1:1024) = matriz_teste_deltmedia1(1:10000,1:1024);
    MT1(1:10000,1025:2048) = matriz_teste_deltmedia2(1:10000,1:1024);
    MT1(1:10000,2049:3072) = matriz_teste_deltmedia3(1:10000,1:1024);


    data = MT1;
    labels = load('test_batch.mat','labels');
```

```
labels = labels.labels;
save('new_test_batch','data','labels');
```

Apêndice D – Rotina para processar imagens aplicando a combinação de LMPs e LMPt

```
% Importar dados e labels treinamento CIFAR-10

S1 = load('data_batch_1.mat');
S2 = load('data_batch_2.mat');
S3 = load('data_batch_3.mat');
S4 = load('data_batch_4.mat');
S5 = load('data_batch_5.mat');

% Montar matriz com dados

S1D = S1.data;
S2D = S2.data;
S3D = S3.data;
S4D = S4.data;
S5D = S5.data;

dados_tr0(1:10000,1:3072) = S1D;
dados_tr0(10001:20000,1:3072) = S2D;
dados_tr0(20001:30000,1:3072) = S3D;
dados_tr0(30001:40000,1:3072) = S4D;
dados_tr0(40001:50000,1:3072) = S5D;


for z = 1:50000
    for k = 1:3
        for i = 1:32
            for j = 1:32
                dados_tr(i,j,k,z) = dados_tr0(z,j+(i-1)*32+(k-1)*1024);
            end
        end
    end
end


% Matriz Preto e Branco

for i = 1:50000
    matt = dados_tr(:,:,:,i);
    treino_bw(:,:,i) = rgb2gray(matt);
end


% Aplicando LMP

for i = 1:50000
    input = padarray(treino_bw(:,:,i),1,'replicate');
    input = padarray(input',1,'replicate');
    input = input';
    matriz_treino_sig(:,:,i) = LMP(input,3, 1);
end
```

```
% Aplicando LMPt

for i = 1:50000
    input = padarray(treino_bw(:,:,i),1,'replicate');
    input = padarray(input',1,'replicate');
    input = input';
    matriz_treino_delt(:,:,i) = LMPt(input,3, 1);
end


% Média dos canais com Matriz LMP

for z = 1:50000
    for k = 1:3
        for j = 1:32
            for i = 1:32
                dd_tr(i,j,z) =
(matriz_treino_sig(i,j,z)*0.5+matriz_treino_delt(i,j,z)*0.5);
                dados_tr(i,j,k,z) = (dados_tr(i,j,k,z) + dd_tr(i,j,z))/2;
            end
        end
    end
end

% Voltando ao formato inicial


for z = 1:50000
    for i = 1:32
            for j = 1:32
                 matriz_treino_sigmedia1(z,(i-1)*32+j) = dados_tr(i,j,1,z);
            end
    end
end

for z = 1:50000
    for i = 1:32
            for j = 1:32
                 matriz_treino_sigmedia2(z,(i-1)*32+j) = dados_tr(i,j,2,z);
            end
    end
end

for z = 1:50000
    for i = 1:32
            for j = 1:32
                 matriz_treino_sigmedia3(z,(i-1)*32+j) = dados_tr(i,j,3,z);
            end
    end
end

    MBD1(1:10000,1:1024) = matriz_treino_sigmedia1(1:10000,1:1024);
    MBD1(1:10000,1025:2048) = matriz_treino_sigmedia2(1:10000,1:1024);
    MBD1(1:10000,2049:3072) = matriz_treino_sigmedia3(1:10000,1:1024);

    MBD2(1:10000,1:1024) = matriz_treino_sigmedia1(10001:20000,1:1024);
    MBD2(1:10000,1025:2048) = matriz_treino_sigmedia2(10001:20000,1:1024);
    MBD2(1:10000,2049:3072) = matriz_treino_sigmedia3(10001:20000,1:1024);
```

```
    MBD3(1:10000,1:1024) = matriz_treino_sigmedia1(20001:30000,1:1024);
    MBD3(1:10000,1025:2048) = matriz_treino_sigmedia2(20001:30000,1:1024);
    MBD3(1:10000,2049:3072) = matriz_treino_sigmedia3(20001:30000,1:1024);

    MBD4(1:10000,1:1024) = matriz_treino_sigmedia1(30001:40000,1:1024);
    MBD4(1:10000,1025:2048) = matriz_treino_sigmedia2(30001:40000,1:1024);
    MBD4(1:10000,2049:3072) = matriz_treino_sigmedia3(30001:40000,1:1024);

    MBD5(1:10000,1:1024) = matriz_treino_sigmedia1(40001:50000,1:1024);
    MBD5(1:10000,1025:2048) = matriz_treino_sigmedia2(40001:50000,1:1024);
    MBD5(1:10000,2049:3072) = matriz_treino_sigmedia3(40001:50000,1:1024);




    data = MBD1;
    labels = load('data_batch_1.mat','labels');
    labels = labels.labels;
    save('new_data_batch_1','data','labels');

    data = MBD2;
    labels = load('data_batch_2.mat','labels');
    labels = labels.labels;
    save('new_data_batch_2','data','labels');

    data = MBD3;
    labels = load('data_batch_3.mat','labels');
    labels = labels.labels;
    save('new_data_batch_3','data','labels');

    data = MBD4;
    labels = load('data_batch_4.mat','labels');
    labels = labels.labels;
    save('new_data_batch_4','data','labels');

    data = MBD5;
    labels = load('data_batch_5.mat','labels');
    labels = labels.labels;
    save('new_data_batch_5','data','labels');

% Importar dados e labels TESTE CIFAR-10

T = load('test_batch.mat');

% Montar matriz com dados

TD = T.data;


dados_te0(1:10000,1:3072) = TD;



for z = 1:10000
    for k = 1:3
        for i = 1:32
            for j = 1:32
                dados_te(i,j,k,z) = dados_te0(z,j+(i-1)*32+(k-1)*1024);
```

```
            end
        end
    end
end


% Matriz Preto e Branco

for i = 1:10000
    matt = dados_te(:,:,:,i);
    teste_bw(:,:,i) = rgb2gray(matt);
end

% Aplicando LMP

for i = 1:10000
    input = padarray(teste_bw(:,:,i),1,'replicate');
    input = padarray(input',1,'replicate');
    input = input';
    matriz_teste_sig(:,:,i) = LMP(input,3, 1);
end

% Aplicando LMPt

for i = 1:10000
    input = padarray(treino_bw(:,:,i),1,'replicate');
    input = padarray(input',1,'replicate');
    input = input';
    matriz_teste_delt(:,:,i) = LMPt(input,3, 1);
end


% Média dos canais com Matriz LMP

for z = 1:10000
    for k = 1:3
        for j = 1:32
            for i = 1:32
                dados_te(i,j,k,z) = (dados_te(i,j,k,z) +
(matriz_teste_sig(i,j,z)*0.5+matriz_teste_delt(i,j,z))*0.5)/2;
            end
        end
    end
end


% Voltando ao formato inicial


for z = 1:10000
    for i = 1:32
            for j = 1:32
                 matriz_teste_sigmedia1(z,(i-1)*32+j) = dados_te(i,j,1,z);
            end
    end
end

for z = 1:10000
```

```
    for i = 1:32
            for j = 1:32
                 matriz_teste_sigmedia2(z,(i-1)*32+j) = dados_te(i,j,2,z);
            end
    end
end

for z = 1:10000
    for i = 1:32
            for j = 1:32
                 matriz_teste_sigmedia3(z,(i-1)*32+j) = dados_te(i,j,3,z);
            end
    end
end

    MT1(1:10000,1:1024) = matriz_teste_sigmedia1(1:10000,1:1024);
    MT1(1:10000,1025:2048) = matriz_teste_sigmedia2(1:10000,1:1024);
    MT1(1:10000,2049:3072) = matriz_teste_sigmedia3(1:10000,1:1024);


    data = MT1;
    labels = load('test_batch.mat','labels');
    labels = labels.labels;
    save('new_test_batch','data','labels');
```

Apêndice E – Rotina para cálculo de LMPs

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LMP - Local Fuzzy Pattern
%
% Input arguments:
% img - Gray scale image.
% neighborSize - number of sample point in an square window.
% beta - pertinence function argument.
%
function result = LMP(img, neighborSize, beta)
if neighborSize < 3 || floor(neighborSize/2) == 0
 error('A vizinhança deve ser um número ímpar maior ou igual a 3!');
end;
img = double(img);
% Tamanho da imagem original
[ysize xsize] = size(img);
if(xsize < neighborSize || ysize < neighborSize)
 error('Imagem muito pequena. Deve ter pelo menos o tamanho da janela.');
end
% weightMatrix = [1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1];
weightMatrix = [1 1 1; 1 1 1; 1 1 1]; %matriz de pesos 1
%weightMatrix = [1 1 1; 1 0 1; 1 1 1]; %matriz de pesos 2
border = fix(neighborSize/2);
dataMatrix = img(2*border : ysize - border, 2*border : xsize - border);
[matrixSizeY matrixSizeX] = size(dataMatrix);
pertinenceSum = zeros(matrixSizeY, matrixSizeX);
weightSum = 0;
for i = 1 : neighborSize
 for j = 1 : neighborSize
 weight = weightMatrix(i, j);
 windowData = img(i : (i+matrixSizeY) - 1, j : (j+matrixSizeX) - 1);

 expData = windowData - dataMatrix;
 pert = 1./(1 + exp(-expData/beta));

 pertinenceSum = pertinenceSum + (pert * weight);
 weightSum = weightSum + weight;
 end;
end;
result = pertinenceSum / weightSum;
```

Apêndice F – Rotina para cálculo de LMPt

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LMP triangular - Local Fuzzy Pattern
%
% Input arguments:
% img - Gray scale image.
% neighborSize - number of sample point in an square window.
% delta - fuzzy number span.
%
function result = LMPt(img, neighborSize, delta)
if neighborSize < 3 || floor(neighborSize/2) == 0
 error('A vizinhança deve ser um número ímpar maior ou igual a 3!');
end;
img = double(img);
% Tamanho da imagem original
[ysize xsize] = size(img);
if(xsize < neighborSize || ysize < neighborSize)
 error('Imagem muito pequena. De ter pelo menos o tamanho da janela.');
end
%weightMatrix = [1 1 1; 1 1 1; 1 1 1]; %matriz de pesos 1
weightMatrix = [1 1 1; 1 0 1; 1 1 1]; %matriz de pesos 2
% weightMatrix = [1 1 1 1 1; 1 1 1 1 1; 1 1 0 1 1; 1 1 1 1 1; 1 1 1 1 1];
border = fix(neighborSize/2);
dataMatrix = img(2*border : ysize - border, 2*border : xsize - border);
[matrixSizeY matrixSizeX] = size(dataMatrix);
pertinenceSum = zeros(matrixSizeY, matrixSizeX);
weightSum = 0;
for i = 1 : neighborSize
 for j = 1 : neighborSize
 weight = weightMatrix(i, j);
 windowData = img(i : (i+matrixSizeY) - 1, j : (j+matrixSizeX) - 1);
 val1 = (windowData - (dataMatrix-delta))/delta;
 val2 = ((dataMatrix+delta) - windowData)/delta;
 pert = max(min(val1, val2), 0);

 pertinenceSum = pertinenceSum + (pert * weight);
 weightSum = weightSum + weight;
 end;
end;
result = pertinenceSum / weightSum;
```