Restauração de Imagens com Redes Neuronais

Relatório de Iniciação Científica

Lucas Antunes Rodrigues

Orientador: Prof. Dr. Magno Teófilo Madeira da Silva

Coorientador: Prof. Dr. Renato Candido

Escola Politécnica

Universidade de São Paulo

13 de Agosto de 2018

Conteúdo

1 Introdução **3**

2 Fundamentação Teórica **3**
2.1 Função de Espalhamento de Ponto 3
2.2 Rede Neuronal Perceptron Multicamadas 4
2.2.1 Estrutura 5
2.2.2 Neurônio 6
2.2.3 Algoritmo *Backpropagation* 10
2.2.4 Exemplos de Aplicação 12
2.3 Rede Neuronal Convolucional 15
2.3.1 Camadas Convolucionais 15
2.3.2 Estrutura 19
2.3.3 Função Custo e Otimizador 20

3 Resultados **20**
3.1 Conjunto de treinamento 21
3.2 Configurações das Redes Neuronais 25
3.2.1 Problema de Quatro Cores 25
3.2.2 Problema de Oito Cores 26
3.3 Resultados das Simulações 28
3.3.1 Problema de Quatro Cores 28
3.3.2 Problema de Oito Cores 32

4 Conclusão **37**

5 Anexos **39**

Referências **39**

Resumo

Este relatório apresenta os resultados obtidos na restauração de imagens por meio de redes neuronais perceptron multicamadas e redes neuronais convolucionais. As imagens em questão estão limitadas a quatro e oito tons de cinza e foram degradadas com uma função de espalhamento de ponto gaussiana.

O relatório está dividido em quatro seções. Na primeira, *Introdução*, são apresentados os conceitos relacionados ao tema do trabalho, além das possíveis aplicações da área de restauração de imagens. Na segunda seção, denominada *Fundamentação Teórica*, é explicada a teoria por trás da degradação das imagens e detalhada a estrutura de uma rede neuronal perceptron multicamadas e uma rede neuronal convolucional. A seguir, na terceira seção, *Resultados*, apresentam-se os resultados obtidos, ilustrando as imagens utilizadas no treinamento das redes e a restauração obtida das imagens testadas. Por fim, na quarta seção, a *Conclusão*, discute-se o desempenho das redes neuronais implementadas, destacando a relevância dos resultados obtidos. Além disso, possíveis continuações do trabalho são expostas.

1 Introdução

A captura de imagens está presente em diferentes atividades, sejam elas científicas, para fins comerciais ou de uso pessoal. Podemos citar exames médicos radiológicos, fotografias astronômicas, fotos em redes sociais, imagens de segurança e outros usos [1, 2]. No entanto, para tais tarefas, um problema sempre a ser enfrentado é a qualidade da imagem obtida. Muitas vezes a imagem adquirida pode apresentar degradações que reduzem sua qualidade, o que prejudica seu uso para o fim desejado [3]. Sendo assim, a restauração de imagens é de grande importância, despertando interesse no meio acadêmico.

Das diferentes maneiras de se enfrentar este problema, o uso de redes neuronais vem demonstrando resultados interessantes, já que lidam bem com a classificação de dados e reconhecimento de padrões [1, 4, 5]. Tais redes são treinadas com um banco de dados rotulados para que sejam capazes de, depois do treinamento, passar por uma fase de testes, em que dados desconhecidos são processados e classificados. Assim, no campo de restauração de imagens, as redes neuronais são capazes de restaurar suficientemente uma imagem, melhorando a qualidade da mesma [2].

Neste relatório, pretende-se expor as atividades exercidas para o uso de redes neuronais perceptron multicamadas (*Multilayer Perceptron* - MLP) e redes neuronais convolucionais (*Convolutional Neural Network* - CNN), fundamentando o conhecimento teórico por trás dessa abordagem e do modelo matemático de uma distorção (*Blur*), apresentando a metodologia exercida e expondo os resultados obtidos.

2 Fundamentação Teórica

Nesta seção, pretende-se apresentar o modelo da distorção de imagens considerada e a teoria para a construção da rede neuronal utilizada na restauração de imagens.

2.1 Função de Espalhamento de Ponto

O modelo do sensor fotográfico utilizado, as condições climáticas de onde se registrou a imagem e outras variantes do ambiente caracterizam a obtenção de uma imagem [3]. Apesar da distinção desses processos de captura, é comum modelar a degradação de uma imagem com uma função de espalhamento de ponto (*Point Spread Function* – PSF) [6].

Na Figura 1, o modelo de distorção linear é apresentado, em que **F** representa a imagem original, **G** a imagem observada e **H** é a distorção (PSF), com

$$\mathbf{G} = \mathbf{F} * \mathbf{H}, \tag{1}$$

em que $*$ significa a convolução em duas dimensões entre a imagem original e a função de distorção. Este cálculo é possível pois o sistema modelado para a degradação da imagem é linear e invariante no espaço de duas dimensões [7], permitindo a convolução entre o sinal de entrada, ou seja, a imagem original **F** e a distorção **H**.

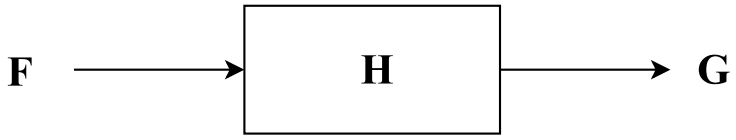


Figura 1: Modelo de distorção.

Como se trata de sinais de duas dimensões, **F**, **G** e **H** são matrizes, sendo n_1 e n_2 os índices relacionados à posição. Cada elemento $\mathbf{F}(n_1, n_2)$ representa um único pixel da imagem original. Seu valor é o número correspondente à cor do pixel numa escala monocromática, também conhecida como nível de cinza. Para uma escala de 8 bits, por exemplo, temos 256 valores possíveis [8]. Já a matriz **H** representa a PSF. O modelo de PSF mais usado na literatura é o gaussiano [9] em que

$$\mathbf{H}(n_1, n_2) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{n_1^2 + n_2^2}{2\sigma^2}\right), \tag{2}$$

sendo σ o desvio padrão. Adota-se o centro da matriz como a origem. Consequentemente, os números de colunas e de linhas da matriz **H** devem ser ímpares, para que a mesma tenha um único elemento central.

Para ilustrar o resultado da distorção causada por essa função de espalhamento de ponto, uma imagem sem distorção é apresentada na Figura 2(a) e as imagens degradadas nas Figuras 2(b) e 2(c), obtidas com $\sigma = 2$ e $\sigma = 5$, respectivamente.

2.2 Rede Neuronal Perceptron Multicamadas

Inspirada na estrutura biológica do cérebro, mais especificamente na maneira como os neurônios se conectam e se comportam, as redes neuronais artificiais são projetadas como modelos computacionais aptos a trabalhar

(a) Original (b) $\sigma = 2$ (c) $\sigma = 5$

Figura 2: Efeito da PSF gaussiana em duas dimensões.

com a classificação de dados e reconhecimento de padrões [2]. Para atingir este fim, tais redes passam por um período de treinamento, em que se conhecem os rótulos dos dados de entrada. No treinamento, busca-se minimizar o erro entre a saída da rede e o rótulo da entada, também conhecido como sinal desejado. Após essa etapa, passa-se à fase de teste, em que dados rotulados não usados no treinamento são apresentados à rede. Dessa forma, verifica-se a capacidade da rede ter aprendido, avaliando sua capacidade de classificação e reconhecimento de padrões.

Na rede MLP, cada nó da rede, também chamado de neurônio [2], nada mais é do que um combinador linear seguido de uma função não linear diferenciável. As conexões entre os nós são ponderadas pelos chamados pesos sinápticos, ou seja, o quanto consideram a informação transmitida por um neurônio anterior ou pelos dados de entrada para um determinado nó da camada seguinte. Já o neurônio tem como cerne a função que lidará com a combinação de suas entradas, também chamada de função de ativação. A seguir, é detalhado cada componente que caracteriza uma rede neuronal MLP.

2.2.1 Estrutura

Para uma rede MLP, os nós são organizados em camadas, sendo que cada uma delas contém um conjunto de neurônios que estão conectados, um a um, com os nós da camada anterior e posterior, se existentes. Vale ressaltar que apenas a camada de entrada não contém neurônios, já que por ela os dados iniciam sua trajetória na rede. As conexões entre os nós são ponderadas por valores chamados de pesos sinápticos. A seguir, o diagrama na Figura 3 ilustra como é a estrutura de uma MLP.

Neste exemplo, a rede contém quatro camadas, sendo a primeira a camada de entrada (identificada por quadrados), as duas do meio as camadas ocultas (internas ou escondidas) e a quarta a camada de saída. Cada circunferência é um neurônio. A primeira camada da rede é composta pelas amostras do sinal de entrada, transmitidas para a primeira camada de neurônios. Conforme o trajeto pelas conexões, chega-se à camada de saída, onde estão os nós responsáveis pela resposta obtida na rede, ou seja, o sinal de saída. Outra forma de representar a estrutura é 3-3-4-1 [2], sendo o primeiro número a quantidade de valores de entrada e os outros a quantidade de neurônios em cada camada.

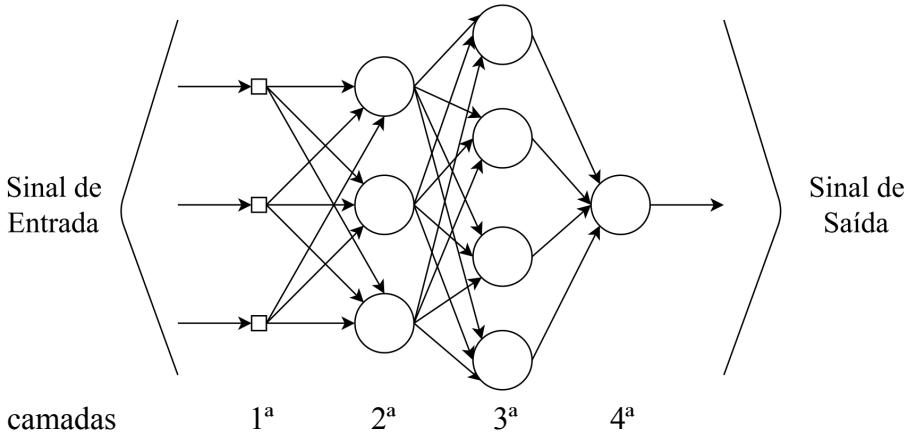


Figura 3: Estrutura 3-3-4-1 de uma rede MLP.

Observa-se que um neurônio de uma determinada camada não se conecta com outro que esteja fora das camadas adjacentes a sua. Por exemplo, um neurônio na segunda camada se conecta apenas com os nós pertencentes à primeira e terceira camada. Além disso, o número de nós por camada é um parâmetro de projeto e depende do problema abordado. Destaca-se também que o número de neurônios na camada de saída está atrelado à quantidade de respostas que se deseja. Sendo assim, caso queira, como exemplo, que a estrutura responda com apenas uma variável, pode-se considerar apenas um neurônio.

2.2.2 Neurônio

De maneira simples, pode-se dizer que um neurônio é um combinador linear atrelado à uma função não linear. Todo neurônio recebe um conjunto de valores de entrada. Esses valores correspondem às saídas dos nós da

camada anterior caso ela exista ou os dados de entrada. Cada valor adquirido é então multiplicado pelo peso da conexão entre o neurônio anterior e o atual, obtendo-se assim uma combinação linear entre a saída dos neurônios anteriores e os pesos sinápticos. Além disso, há um peso somado a essa conta chamado de *bias*, independente das saídas de qualquer neurônio. O resultado desta operação é então enviado como variável de entrada para a função não linear, conhecida como função de ativação. Por fim, a saída desta função será a resposta final do nó. Este processo ocorre igualmente para todos os outros nós da rede.

Para resumir o procedimento, temos que a saída y de um neurônio j em uma determinada camada l é

$$y_j^{(l)} = \varphi_j^{(l)}(v_j^{(l)}), \tag{3}$$

sendo que

$$v_j^{(l)} = \sum_{i=1}^{N_{l-1}} w_{i,j}^{(l)} y_i^{(l-1)} + b_j^{(l)}, \tag{4}$$

com N_{l-1} representando a quantidade de nós da camada anterior conectados ao neurônio j, $b_j^{(l)}$ o valor referido como *bias* e $w_{i,j}^{(l)}$ o peso entre o nó i, da camada anterior, e o neurônio j, da camada atual. Na equação, $\varphi_j^{(l)}(.)$ representa a função de ativação do neurônio. Vale destacar que, no caso da camada de entrada, os valores y_i são na verdade os dados de entrada. Na Figura 4, ilustra-se o cálculo apresentado.

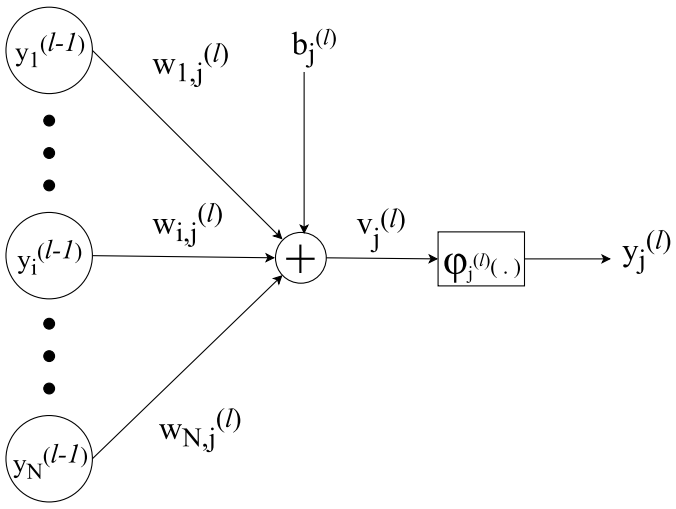


Figura 4: Diagrama de um neurônio.

A função de ativação $\varphi(.)$, responsável pela não linearidade do sistema, desempenha um papel importante na rede, já que delimita as possíveis respostas de um neurônio. Dentre as funções comumente utilizadas, temos a função sigmoidal [2]. Tal função é definida como

$$\varphi_j^{(l)}(v_j^{(l)}) = \frac{1}{1 + e^{-\lambda v_j^{(l)}}}, \tag{5}$$

para todo $v_j^{(l)} \in \mathbb{R}$, em que λ determina o quão abrupto será a curva. Essa função mapeia a reta real no intervalo $[0, 1]$, portanto, a saída do neurônio fica limitada a estes valores. Ainda, para $v_j^{(l)} \to \infty$, $\varphi_j^{(l)}(v_j^{(l)}) = 1$ e para $v_j^{(l)} \to -\infty$, $\varphi_j^{(l)}(v_j^{(l)}) = 0$, sendo assim, caso a resposta desejada seja booleana, a função sigmóide é uma ótima candidata para os neurônios de saída da rede. Tais características são interessantes para estabelecer se um determinado neurônio j tem sua saída y significativa para a rede ou não, conforme o valor obtido da função de ativação $\varphi(.)$[2]. Além disso, para que o algoritmo de aprendizado funcione, é necessário que a função de ativação seja diferenciável.

Com o objetivo de ilustrar a função sigmoidal, na Figura 5 temos sua curva para $\lambda = 1$.

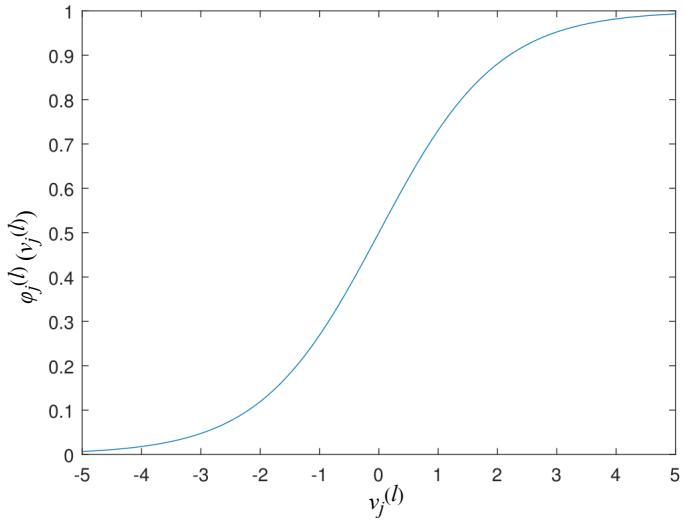


Figura 5: Curva da função sigmoide, com $\lambda = 1$.

Outra função comumente utilizada na literatura [2, 5] é a unidade linear retificada (*Rectified Linear Unit* - ReLU), definida como

$$\varphi_j^{(l)}(v_j^{(l)}) = max(0, v_j^{(l)}), \tag{6}$$

para todo $v_j^{(l)} \in \mathbb{R}$. Essa função mapeia a reta real no intervalo $[0, \infty]$ e é comumente utilizada nas camadas internas de uma rede neuronal. Sua grande vantagem se dá pelo fato de diminuir a chance de haver saturação na saída dos neurônios, já que para valores positivos não há um valor limitante para $\varphi_j^{(l)}(v_j^{(l)})$. Assim, diferentemente da sigmoidal, não há saturação em ambas as direções da reta real.

Com o objetivo de ilustrar a função, temos sua curva na Figura 6.

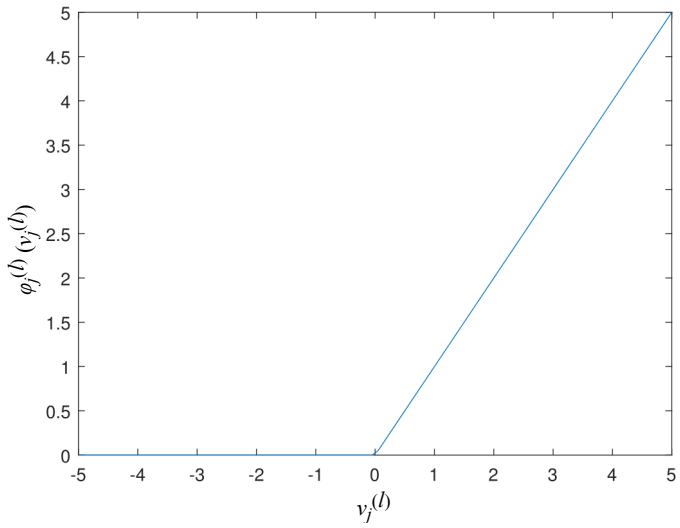


Figura 6: Curva da função ReLU.

Por fim, neste trabalho, foi utilizada também a função de ativação *softmax* definida como

$$\varphi_j^{(l)}(v_j^{(l)}) = \frac{e^{v_j^{(l)}}}{\sum_{i=1}^{N} e^{v_i^{(l)}}}, \tag{7}$$

para todo x $\in \mathbb{R}$. Nesta equação, $v_i^{(l)}$ representa as entradas das funções de ativação dos N neurônios da mesma camada. Dessa forma, a saída de um neurônio com a função de ativação *softmax* depende da entrada da função de ativação de todos os outros neurônios da mesma camada. Vale ressaltar que esta função é uma generalização da função sigmoidal para o caso de haver a classificação de dados em mais que duas categorias. Consequentemente, a função *softmax* é comumente utilizada na camada de saída. Além disso, seus valores estão definidos em $0 \leq \varphi_j^{(l)}(v_j^{(l)}) \leq 1$.

2.2.3 Algoritmo *Backpropagation*

Para que a rede MLP aprenda a responder conforme o desejado, há uma etapa de treinamento, sendo o algoritmo de retropropagação (*Backpropagation*) o mais comumente utilizado para essa tarefa. Esse algoritmo tem como objetivo atualizar os pesos sinápticos de cada conexão da rede para que se obtenham os valores desejados. O processo ocorre graças ao treinamento supervisionado da rede, já que a saída é comparada com o sinal desejado, obtendo um sinal de erro que deve ser minimizado segundo algum critério.

O algoritmo foi obtido a partir da minimização da função custo definida pela equação

$$\zeta(n) = \frac{1}{2}\sum_{j=1}^{N} |e_j(n)|^2, \tag{8}$$

sendo N o número de neurônios da camada de saída da rede e $e_j(n)$ o erro calculado para cada neurônio j dessa camada, comparando-se a saída $y_j^{(L)}(n)$ com o sinal desejado $d_j(n)$ na iteração n. É possível descrever o funcionamento do algoritmo em quatro passos, enumerados a seguir [2].

1. Inicialização

   Os pesos sinápticos e o *bias* são inicializados com números aleatórios gerados a partir da distribuição normal com média $\mu = 0$ e desvio padrão $\sigma = 1$.

2. Cálculo progressivo

   As saídas de cada neurônio da camada l devem ser calculadas como descrito na Equação (3). Em seguida, calcula-se o sinal de erro do neurônio j da camada de saída ($l = L$) como

   $$e_j(n) = d_j(n) - y_j^{(L)}(n), \tag{9}$$

3. Cálculo regressivo

   Como se deseja a minimização da função custo definida em (8), temos que sua derivada com relação a $w_{i,j}^{(l)}(n)$ pode ser escrita como

   $$\frac{\partial \zeta(n)}{\partial w_{i,j}^{(l)}(n)} = -\delta_j^{(l)}(n) y_i^{(l-1)}(n) \tag{10}$$

   sendo que $\delta_j^{(l)}(n)$ é definido em [2] como gradiente local e deve ser calculado como explicado a seguir. Para um neurônio j da camada de

saída ($l = L$), deve-se utilizar

$$\delta_j^{(L)}(n) = e_j^{(L)}(n)\varphi'(v_j^{(L)}(n)), \tag{11}$$

sendo $\varphi'(.)$ a derivada da função de ativação e $v_j^{(L)}$ definido como na Equação (4). Já para um neurônio j de uma camada escondida $l \neq L$, a expressão é dada por

$$\delta_j^{(l)}(n) = \varphi'(v_j^{(l)}(n)) \sum_i \delta_i^{(l+1)}(n) w_{i,j}^{(l+1)}(n). \tag{12}$$

A partir do gradiente local, atualizam-se os pesos com a equação

$$w_{i,j}^{(l)}(n+1) = w_{i,j}^{(l)}(n) + \alpha[w_{i,j}^{(l)}(n) - w_{i,j}^{(l)}(n-1)] - \eta \frac{\partial \zeta(n)}{\partial w_{i,j}^{(l)}(n)}, \tag{13}$$

sendo α o *momentum*, escolhido no intervalo [0,1] e usado para aumentar a velocidade de convergência do algoritmo. Já η é a taxa de aprendizado. Esse método de atualização dos pesos é conhecido como gradiente descendente estocástico.

4. Atualizando os pesos, deve-se voltar ao passo 2 até que o erro mínimo pré-especificado seja atingido.

Detalhes da dedução do algoritmo *Backpropagation* podem ser encontrados em [2].

Além disso, vale ressaltar que para prolongar o treinamento da rede, é comum aplicar o algoritmo *Backpropagation* para toda a coleção de entradas mais de uma vez. O treinamento realizado com a coleção completa dos dados de entrada é chamado de época. Em geral, o algoritmo pode levar várias épocas até convergir. Cabe observar ainda que antes de cada época, os dados de entrada são misturados para gerar diversidade entre épocas.

Para cada vez que o erro obtido pela rede é calculado para uma única entrada, nomeia-se iteração. É possível que a atualização dos pesos ocorra apenas após mais de uma iteração da rede. Assim, o valor utilizado para o cálculo do gradiente da função custo em relação aos pesos é

$$\zeta(n) = \frac{1}{2k} \sum_{i=1}^{k} \sum_{j=1}^{N} |e_j(n)|^2, \tag{14}$$

sendo k o número de iterações considerado. Caso $k = 1$, nomeia-se o método de estocástico. Para $1 < k < m$, sendo m o número de amostras

total, nomeia-se o método de *mini-batch*. Por fim, para $k = m$, nomeia-se o método de *batch*. É comum na literatura o uso do *mini-batch*, já que, para k iterações, com $1 < k < m$, a precisão do gradiente para alcançar um mínimo da função custo é maior. Comparado com o *batch*, utilizar k iterações também se demonstra melhor, já que o custo computacional para cada atualização dos pesos é menor.

Para ilustrar a diferença entre os três métodos, a Figura 7, disponível em http://deeplearning.buzz/2017/06/01/what-is-batch-size-and-epoch-in - neural-network/, mostra a trajetória de dois coeficientes da rede até atingir o mínimo da função custo, considerando os três métodos. Vale ressaltar a diferença de variância entre os três métodos.

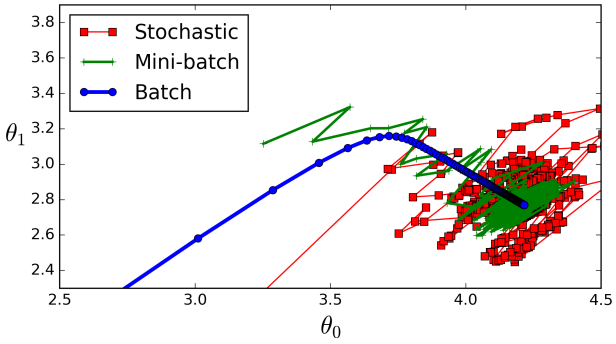


Figura 7: Comparação entre *batch*, *mini-batch* e estocástico.

2.2.4 Exemplos de Aplicação

Com o intuito de implementar uma rede MLP, foram considerados dois problemas, o primeiro o problema XOR e o segundo o problema das meia luas (*double-moon*).

No caso do XOR, a rede neuronal projetada deve aprender com as entradas propostas a se comportar como uma porta lógica XOR. As entradas do sistema são compostas por pares de números, todos binários, como os pares (p,q) da Tabela 1. Já os valores desejados $d_j(n)$ são determinados pela porta XOR, que está apresentada a seguir, sendo $\oplus$ o símbolo da operação. Como exemplo, caso a rede receba como entrada o par (0,1), deve retornar a saída $y = 1$.

Como solução, foi proposta uma rede composta por quatro camadas, contendo dois neurônios na primeira camada escondida, dois na segunda e apenas um na camada de saída, além da camada de entradas. Simboliza-se essa composição como 2-2-2-1. A função de ativação utilizada foi a sigmoide. Já a

Tabela 1: Tabela Verdade XOR

p	q	p⊕q
0	0	0
0	1	1
1	0	1
1	1	0

taxa de aprendizado foi mantida constante $\eta = 1$ e o momentum também, $\alpha = 0{,}1$. Os pesos $w_{i,j}$ iniciais da rede e o peso do bias b_j foram gerados aleatoriamente a partir da distribuição normal com média $\mu = 0$ e desvio padrão $\sigma = 1$. Foi utilizado o método estocástico para o treinamento.

Para ilustrar a solução, na Figura 8 é possível observar um gráfico onde as retas que dividem a região em A e B representam a curva obtida pela rede. Para os pares de pontos pertencentes à região A, a rede responde como 0, já os pares presentes na região B são classificados como 1. Logo, os círculos pretos, que representam os pares (0,1) e (1,0), ou seja, as possíveis entradas do sistema, são classificados corretamente. Já os pares (0,0) e (1,1) são também classificados como desejado, já que se encontram na região A.

Para que o processo de decisão ocorra, há um decisor que interpreta a saída da rede e delimita a resposta em 0 ou 1, saída discreta. Isso se deve ao fato de que a rede tem como possíveis entradas e saídas o intervalo contínuo entre 0 ou 1. Assim, é viável gerar a fronteira de classificação da rede, representada pelas retas na Figura 8.

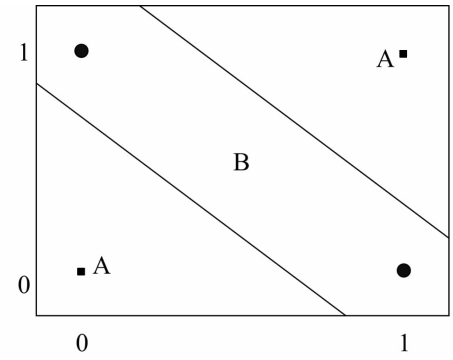


Figura 8: Gráfico de saída para o problema XOR.

Já na Figura 9, é apresentada a curva do erro quadrático da saída da rede ao longo das iterações. Observa-se que a partir do instante $n = 600$, o erro já foi consideravelmente reduzido, indicando que a rede foi capaz de modelar a operação XOR adequadamente. Antes desse período, a partir da curva do erro é possível observar que o aprendizado ainda não ocorreu, simbolizando que a rede estava despreparada para o problema.

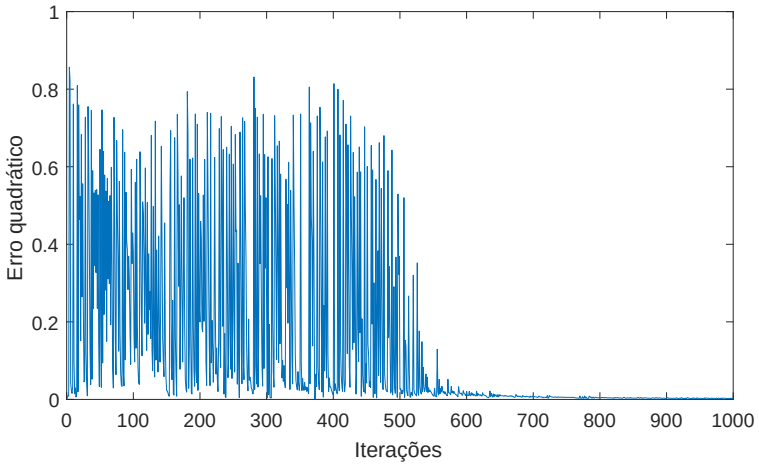


Figura 9: Gráfico da função erro da rede.

No problema das meia luas (*double-moon*), a rede neuronal deve aprender a classificar pontos pertencentes a duas meia luas, distinguindo em qual das duas cada ponto pertence. Este problema é determinado por três variáveis que caracterizam a dificuldade de uma rede MLP solucionar a classificação. A primeira variável é o raio r das meia luas. Em segundo lugar, temos a largura l das mesmas. Então, como última variável, a distância d entre elas. Para ilustrar o desafio, as duas meia luas se encontram na Figura 10, com os três parâmetros destacados.

Para o treinamento da rede neuronal, pares de pontos (x,y) pertencentes a cada uma das meia luas constituem as amostras de entrada. Assim, os valores desejados são binários, já que a saída deve dizer se os dados de entrada pertencem ao grupo de pontos da primeira meia lua ou da segunda. Como a saída requer apenas uma variável, um único neurônio nessa camada é suficiente.

Como solução do problema, em que $l = 6$, $d = 4$ e $r = 10$, foi proposta uma rede composta por três camadas, contendo vinte neurônios na primeira camada escondida e um na segunda, ou seja, 2-20-1, além da camada de entradas. A função de ativação utilizada foi a sigmoide. Já a taxa de aprendizado foi reduzida linearmente de $\eta = 1$ até $\eta = 0{,}00001$ ao longo das iterações,

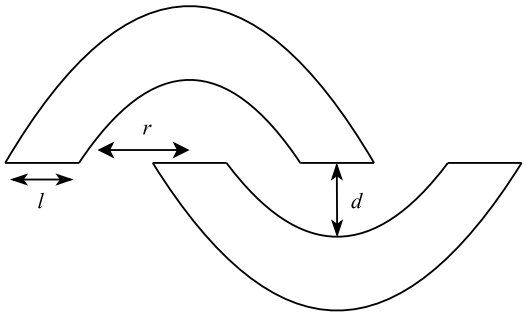


Figura 10: Ilustração das meia luas.

enquanto o momentum foi mantido nulo, $\alpha = 0$. Os pesos $w_{i,j}$ iniciais da rede e o bias b_j foram gerados aleatoriamente a partir da distribuição normal de média $\mu = 0$ e desvio padrão $\sigma = 1$. Foi utilizado o método estocástico para o treinamento.

2.3 Rede Neuronal Convolucional

As redes CNN são redes especializadas no processamento e classificação de dados organizados em matrizes [2, 5]. Nesta seção, descreve-se cada componente que caracteriza uma CNN. Primeiramente, a operação convolução para o domínio discreto é detalhada, para que uma camada convolucional seja, em seguida, caracterizada. Em segundo lugar, a estrutura geral de uma CNN é apresentada. Por fim, a função custo de entropia cruzada e o otimizador *Adam* também são descritos.

2.3.1 Camadas Convolucionais

Considere duas matrizes **K** e **I**. A convolução em duas dimensões dessas matrizes é definida como

$$S(i,j) = (K * I)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(i-m, j-n)K(m,n), \qquad (15)$$

sendo i e j índices da matriz resultante S e m e n índices das somatórias [5]. Esse tipo de operação caracteriza uma camada convolucional da CNN, mas as matrizes são de dimensão finita, o que faz com que os somatórios em (15)

sejam finitos. Além disso, a matriz **K** em geral é chamada de filtro e **I** é uma entrada que pode ser uma imagem ou a saída de uma camada convolucional. Considerando que **K** (filtro) é uma matriz 2×2 e **I** (entrada) é uma matriz 3×4, a operação de convolução está ilustrada na Figura 11.

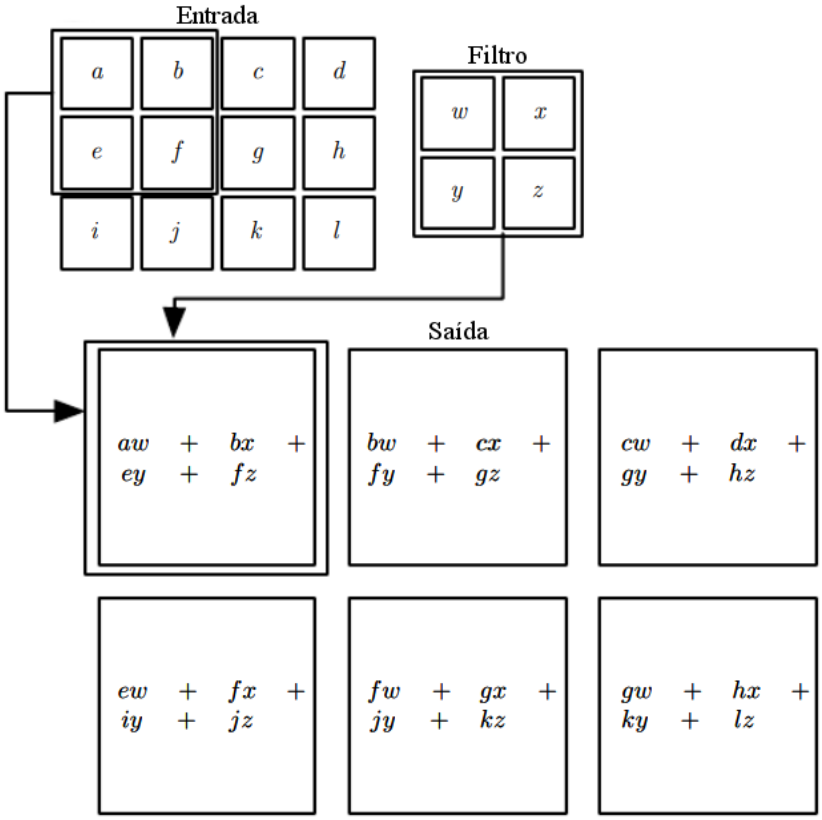


Figura 11: Um exemplo de convolução bidimensional entre um filtro e uma matriz de entrada. As setas indicam o resultado da operação para o índice $i = 1$ e $j = 1$ da saída.

Em uma camada convolucional, é comum considerar mais de um filtro. Para cada filtro, calcula-se a convolução em duas dimensões da matriz do filtro com a matriz de entrada. A esse resultado, soma-se o bias do filtro. A matriz resultante entra então em uma função de ativação não linear e uma das matrizes da saída é obtida. A saída terá uma dimensão a mais que corresponde ao número de filtros da camada. Cabe observar que é comum na literatura considerar filtros quadrados e de mesmo tamanho para uma mesma camada [2, 5]. A consequência desse processo é que as dimensões

da saída de uma camada convolucional serão diferentes da entrada. Por exemplo, para k filtros $f \times f$ e uma matriz de entrada $a \times b$, a saída será do tipo $(a - f + 1) \times (b - f + 1) \times k$. Com o objetivo de ilustrar esse processo, na Figura 12 é possível visualizar a entrada e saída da convolução de uma determinada camada para uma entrada 4×4 e 4 filtros 3×3.

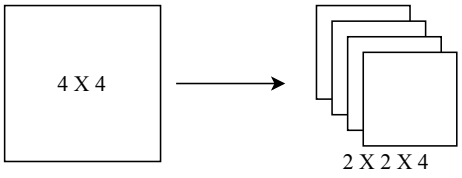


Figura 12: Entrada e saída de uma camada convolucional para 4 filtros 3×3.

Os filtros das camadas convolucionais são matrizes de pesos. Cada elemento da matriz representa um único peso. Assim, para 4 filtros 3×3, como visto na Figura 12, existem $3 \times 3 \times 4 + 4 = 40$ pesos, já que o *bias* de cada filtro está incluso nesse cálculo. Após a convolução ocorrer, é somado a cada elemento da saída o *bias*, valor igual dado um mesmo filtro. Em seguida, a função de ativação é aplicada para cada elemento das matrizes resultantes. Assim, os elementos da saída da camada convolucional serão resultados da convolução com os filtros, soma do *bias* e uso da função de ativação. Para ilustrar o processo, dado uma função de ativação $\varphi(.)$ e o *bias* k do quarto filtro, a saída final da camada convolucional para o exemplo da Figura 12 é ilustrada na Figura 13.

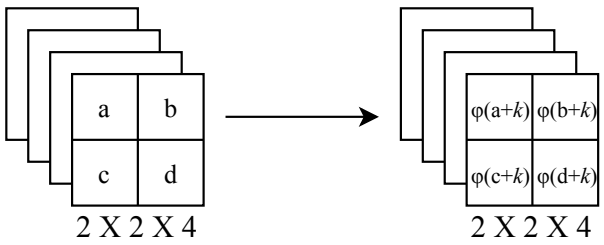


Figura 13: Saída da camada convolucional dada a função de ativação φ e o *bias* k do quarto filtro.

Vale ressaltar que, caso a entrada de uma camada convolucional seja de dimensão maior do que 2, como exemplo a saída $2 \times 2 \times 4$ da Figura 13, os filtros desta camada terão sua terceira dimensão igual à da entrada. A convolução será então tridimensional. Para ilustrar o cálculo, na Figura 14

se encontra o diagrama para a convolução tridimensional. Dado uma entrada $a \times b \times c$ e k filtros $f \times f \times c$, a saída será $(a - f + 1) \times (b - f + 1) \times k$, conforme ilustrado no exemplo da Figura 15. Neste exemplo, encontram-se 6 filtros $3 \times 3 \times 4$, totalizando $3 \times 3 \times 4 + 6 = 42$ pesos, para uma entrada $4 \times 4 \times 4$ e saída $2 \times 2 \times 6$.

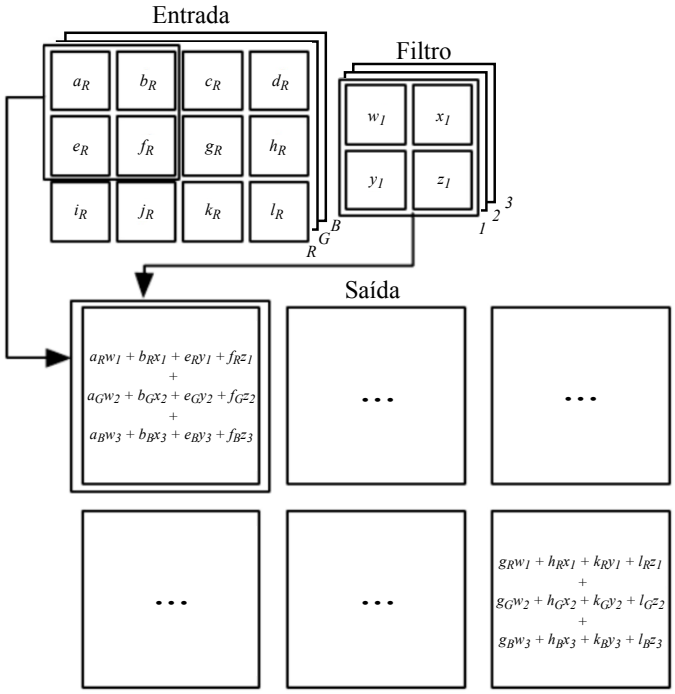


Figura 14: Um exemplo de convolução tridimensional.

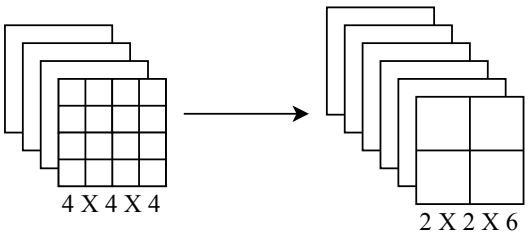


Figura 15: Entrada e saída de uma camada para 6 filtros $3 \times 3 \times 4$.

Para cada iteração do algoritmo *Backpropagation*, os pesos dos filtros são atualizados, assim como ocorre para os pesos de um neurônio na rede MLP. O algoritmo *Backpropagation* para uma camada convolucional apresenta al-

gumas mudanças se comparado com o já descrito neste relatório para redes MLP. Detalhes da dedução do algoritmo podem ser encontrados em [2, 5].

2.3.2 Estrutura

Para uma rede CNN, a estrutura se divide em camadas convolucionais e camadas de nós como já visto numa rede MLP. Primeiramente, temos a camada de entrada sendo uma matriz de dados. Em seguida, são postas camadas convolucionais com o objetivo de reduzir o tamanho das dimensões da saída de dados a cada camada que se passa. Por fim, camadas de neurônios recebem a saída da última camada convolucional em forma de vetor e retornam a saída final desejada para a rede.

Como já detalhado na seção anterior, a saída de uma camada convolucional l tem o tamanho $(a^{(l-1)} - f^{(l)} + 1) \times (b^{(l-1)} - f^{(l)} + 1) \times k^{(l)}$, sendo a e b dimensões da saída da camada $l - 1$. A primeira e a segunda dimensão da saída de dados reduzem de acordo com $f^{(l)}$ e a terceira dimensão é igual a quantidade de filtros $k^{(l)}$. É comum na literatura o aumento da quantidade de filtros k a cada camada convolucional que se passa [2, 5]. Assim, a tendência é que, após certa quantidade de camadas convolucionais, a saída de dados seja $1 \times 1 \times k^{(C)}$, sendo $k^{(C)}$ o número de filtros da última camada convolucional C da rede. Como a saída de dados passa a ser um vetor, é possível utiliza-la como entrada das camadas de neurônios já vistas em redes MLP. Por fim, segue-se a estrutura já apresentada nas redes MLP, construindo uma ou mais camadas de neurônios que categorizam os dados conforme a quantidade de saídas desejada.

Para ilustrar a estrutura de uma CNN, o diagrama na Figura 16 apresenta uma rede com uma camada de entrada, três camadas convolucionais de filtros 3×3 e um camada de neurônios. A quantidade de filtros aumentou a cada camada, sendo 1, 2 e 4, respectivamente. A entrada é de tamanho $7 \times 7 \times 1$ e a saída da rede é dada pelos quatro neurônios na última camada.

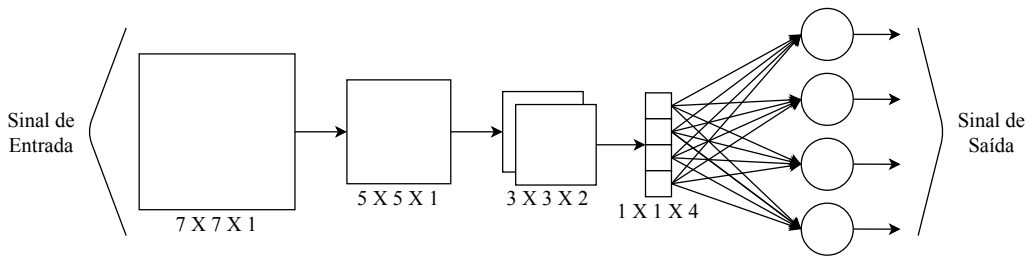


Figura 16: Exemplo de estrutura de uma rede CNN.

2.3.3 Função Custo e Otimizador

Além da função custo do erro quadrático, outra função frequentemente utilizada na literatura [2, 5] é a entropia cruzada, dada por

$$\zeta(k) = -\sum_{j=1}^{N} d_j(k) \ln(y_j(k)), \tag{16}$$

em que, para a iteração k da rede, N denota o número de neurônios da camada de saída da rede, $d_j(k)$ a saída esperada da rede (sinal desejado) e $y_j(k)$ a saída de cada neurônio j, com $j = 1,2,\ldots,N$. Tal função custo é normalmente empregada quando se deseja atribuir uma classificação dentre N categorias possíveis para uma certa entrada. Neste relatório, a função custo de entropia cruzada foi utilizada tanto para redes MLP quanto as CNN. Para maiores detalhes, ver a seção de resultados.

Por fim, além do otimizador gradiente descendente estocástico, outro algoritmo utilizado neste trabalho foi o Adam (nome derivado de *adaptive moment estimation*). Para atualizar os pesos da rede, esse método utiliza o decaimento exponencial médio dos gradientes passados ao quadrado e o decaimento exponencial médio dos gradientes passados. Assim, a atualização dos pesos para o método Adam é dada por

$$w(n+1) = w(n) - \frac{\eta}{\sqrt{b(n)} + \epsilon} a(n), \tag{17}$$

sendo $w(n)$ um certo peso para a iteração n, η a taxa de aprendizado e ϵ um ajuste constante usado para evitar divisão por zero, valendo, por exemplo, 10^{-8}. Já $a(n)$ e $b(n)$, representam os decaimentos médios já mencionados dados por

$$a(n) = \frac{\beta_1 a(n-1) + (1-\beta_1)\delta(n)}{1-\beta_1}, \tag{18}$$

$$b(n) = \frac{\beta_2 b(n-1) + (1-\beta_2)\delta^2(n)}{1-\beta_2}, \tag{19}$$

sendo $\delta(n)$ o gradiente da função custo em relação ao peso $w(n)$. A literatura [10] propõe valores padrões de 0,9 e 0,999 para os parâmetros β_1 e β_2, respectivamente. Assim, para o otimizador Adam, o único parâmetro normalmente ajustado de acordo com cada aplicação é a taxa de aprendizado η.

3 Resultados

Nesta seção, são apresentados os resultados obtidos ao restaurar imagens de quatro e oito cores numa escala de níveis de cinza. Foram testadas diversas

redes MLP e CNN a fim de otimizar todos os parâmetros. Para quatro cores, a rede escolhida foi a que levou aos melhores resultados observados em termos de desempenho e tempo de simulação. O mesmo vale para as quatro redes escolhidas no caso de oito cores.

Vale ressaltar que os resultados para o problema de quatro cores foram simulados no *software* MATLAB. A rede utilizada foi inteiramente programada na linguagem do MATLAB sem utilizar pacotes específicos para isso, visando o aprendizado da estrutura das redes neuronais. Em seguida, a mesma rede foi programa em Python. Quando os resultados obtidos coincidiram com os do MATLAB, decidiu-se trabalhar apenas com Python. Para o problema de oito cores, os resultados obtidos foram simulados a partir da biblioteca *Keras*. Um trecho do código para uma das redes CNN utilizando o *Keras* está em anexo ao final do relatório.

3.1 Conjunto de treinamento

Para o treinamento das redes neuronais no problema de quatro cores, foram selecionadas dez fotografias contendo 256×256 pixels em escala de cinza com 8 bits por pixel. Já para o problema de oito cores, foram selecionadas 19 fotografias com as mesmas características.

Antes da degradação ser aplicada, as imagens tiveram sua resolução de cores reduzida, adequando-se para quatro ou oito níveis de cinza. É possível observar essa transformação na Figura 17. Na Figura 17(a) temos a imagem original de 256 tons de cinza e na Figura 17(b) é possível observar o efeito da redução.

(a) 256 cores (8 bits/pixel) (b) 4 cores (2 bits/pixel) (c) 8 cores (4 bits/pixel)

Figura 17: Redução de niveís de cinza das imagens.

Após a redução de cores, as imagens selecionadas foram degradadas com a aplicação da distorção gaussiana bidimensional, descrita pela Equação (2)

da página 4. Com o objetivo de restaurar as imagens com diferentes níveis de degradação para o caso de quatro cores, foram considerados cinco valores de desvio padrão da PSF gaussiana, ou seja, $\sigma = 0{,}5$, 1, 1,5, 2 e 3. Já o caso de oito cores, visando estudar a melhor rede para a solução do problema, fixou-se um único valor de desvio padrão da PSF gaussiana, $\sigma = 2$.

Como exemplo, as dez imagens originais usadas no treinamento de quatro cores são mostradas na Figura 18(a) e as imagens degradadas considerando a PSF gaussiana com $\sigma = 2$ são mostradas na Figura 18(b).

(a) Imagens originais.

(b) Imagens degradadas com PSF gaussiana 7×7 e $\sigma = 2$.

Figura 18: Coleção de imagens para o treinamento da rede.

Para o caso de oito cores, as dezenove imagens originais estão expostas na Figura 19 e as imagens degradadas são mostradas na Figura 20.

Figura 19: Imagens originais.

Figura 20: Imagens degradadas com PSF gaussiana 7×7 e $\sigma = 2$.

3.2 Configurações das Redes Neuronais

A seguir, são apresentadas as diferentes redes neuronais utilizadas para o problema de quatro e oito cores.

3.2.1 Problema de Quatro Cores

No problema em questão, decidiu-se utilizar uma rede MLP de quatro camadas, tendo primeiramente 49 valores de entrada, quarenta neurônios na segunda e terceira camadas e quatro neurônios na camada de saída, ou seja, 49-40-40-4. Tal estrutura visa minimizar o número de camadas da rede, mantendo uma elevada quantidade de neurônios e conexões se comparado com as redes utilizadas nos exemplos de aplicação da Seção 2.2.4. A minimização do número de camadas é importante para garantir a adaptação feita pelo algoritmo *Backpropagation*, evitando que os pesos da rede fiquem estagnados.

Todos os neurônios da rede tiveram como função de ativação a função sigmóide, dada pela Equação (5). Assim, os valores de saída de todos os nós ficam entre 0 e 1. Já a taxa de aprendizado foi configurada para reduzir linearmente de $\eta = 0{,}1$ a $\eta = 0{,}001$ conforme as iterações da rede. O *momentum* foi mantido constante em $\alpha = 0{,}001$, igual ao menor valor assumido pela taxa de aprendizado.

Como entrada da rede, considerou-se em cada iteração uma matriz 7×7 de pixels de uma das dez imagens degradadas, com o pixel a ser estimado em sua posição central ($n_1 = n_2 = 4$). Vale ressaltar que os valores dos elementos da matriz são os números assumidos pelos pixels numa escala de níveis de cinza. A seleção do pixel foi considerada aleatória entre as dez imagens, o que levou a 655360 ($10 * 256 * 256$) iterações, uma para cada pixel alvo. O treinamento é completado após quatro ocorrências deste processo, ou seja, quatro épocas. O esquema dos passos descritos está na Figura 21.

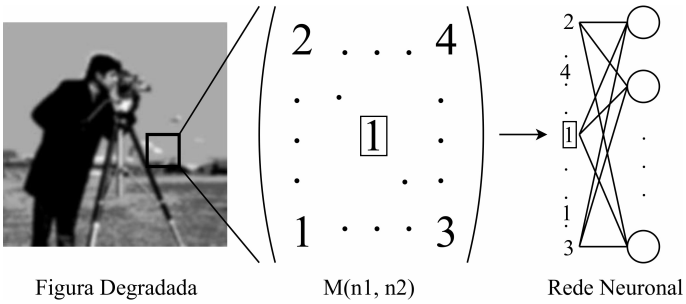


Figura 21: Passos realizados para a entrada da rede MLP.

Na camada de saída, foram considerados quatro neurônios, um para cada nível de cinza possível. Após um algoritmo de decisão, seleciona-se o neurônio que apresenta o maior valor de saída. Por exemplo, caso a saída da rede seja a sequência de valores (0,2; 0,1; 0,3; 0.9), o quarto neurônio, de saída 0,9, será selecionado e a cor branca será a escolhida pela rede, nível correspondente à quarta cor possível entre os quatro níveis do problema.

3.2.2 Problema de Oito Cores

No problema em questão, decidiu-se utilizar duas redes MLP e duas redes CNN diferentes entre sí. No caso anterior, para entrada das redes, considerou-se em cada iteração uma matriz 7×7 de pixels de uma das 19 imagens degradadas, com o pixel a ser estimado na posição central ($n_1 = n_2 = 4$). A seleção do pixel foi aleatória entre as 19 imagens, o que levou a 1.245.184 ($19 * 256 * 256$) iterações, uma para cada pixel alvo. O esquema dos passos para as redes CNN está na Figura 22.

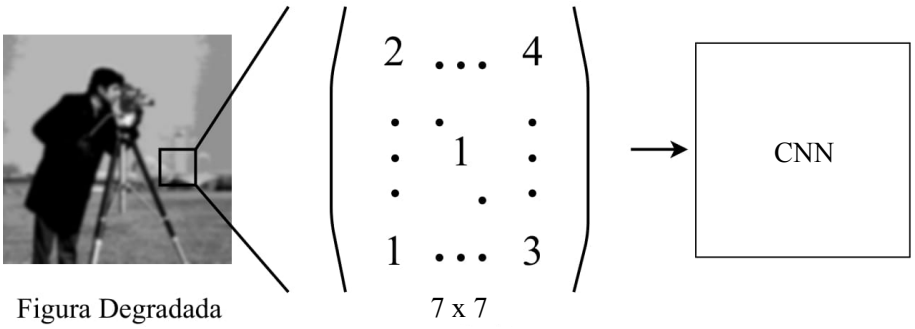


Figura 22: Passos realizados para a entrada das CNNs.

As estruturas das quatro redes são apresentadas nos diagramas das Figuras 23 e 24. Primeiramente, as redes MLP são descritas. Vale ressaltar que a única diferença entra elas é o otimizador. Em seguida, as redes CNN são detalhadas. Neste caso, a diferença está no tamanho da rede. As duas redes CNN utilizaram o otimizador Adam, já que o SGD não convergiu, não obtendo resultado adequado após treinamento.

Na camada de saída de todas as redes, foram considerados oito neurônios, um para cada nível de cinza. Após um algoritmo de decisão, seleciona-se o neurônio que apresenta o maior valor de saída. Por exemplo, caso a saída da rede seja a sequência (0,2; 0,1; 0,3; 0,9; 0,4; 0,0; 0,1; 0,2), o quarto neurônio, de saída 0,9, será selecionado e sua respectiva cor será a escolhida pela rede.

	MLP-SGD	**MLP-Adam**
Camada de entrada	Matriz 7x7 transformada em um vetor de 49 elementos.	
1ª Camada oculta	40 neurônios, função de ativação ReLU.	
2ª Camada oculta	40 neurônios, função de ativação ReLU.	
Camada de saída	8 neurônios, função de ativação *Softmax*.	
Otimizador	SGD $\eta = 0{,}001$, $\alpha = 0{,}001$, sem decaimento, *mini-batch*, k = 32.	Adam $\eta = 0{,}001$, sem decaimento, *mini-batch*, k = 32.
Função Custo	Entropia Cruzada.	

Figura 23: Estrutura das redes MLP.

	CNN-Maior	**CNN-Menor**
Camada de entrada	Matriz 7x7.	
1ª Camada oculta	8 filtros 3x3x1, função de ativação ReLU.	8 filtros 3x3x1, função de ativação ReLU.
2ª Camada oculta	8 filtros 3x3x8, função de ativação ReLU.	16 filtros 3x3x8, função de ativação ReLU.
3ª Camada oculta	8 filtros 3x3x8, função de ativação ReLU.	32 filtros 3x3x16, função de ativação ReLU.
Camada de saída	8 neurônios, função de ativação *Softmax.*	
Otimizador	Adam $\eta = 0{,}001$, sem decaimento, *mini-batch*, k = 32.	
Função Custo	Entropia cruzada.	

Figura 24: Estrutura das redes CNN.

As quatro redes utilizaram as mesmas funções de ativação e função custo. Isso se deve ao fato de obterem desempenhos melhores se comparado ao uso da função de ativação sigmoide e a função custo do erro quadrático [2, 5]. Optou-se também por manter a mesma taxa de aprendizado para todas as redes.

Para as duas redes MLP, optou-se por manter a mesma estrutura já que foi a de melhor desempenho dentre os testes realizados e é baseada na rede utilizada para o problema de quatro cores. Além disso, desejou-se comparar os dois otimizadores, observando as restaurações obtidas e o tempo levado para o treinamento das duas redes.

Para as duas redes CNN, optou-se por comparar duas redes com quantidades de neurônios diferentes ao longo das camadas, observando o efeito dessa diferença nos resultados obtidos e no custo computacional.

3.3 Resultados das Simulações

A seguir, são apresentadas as restaurações obtidas para os problemas de quatro e oito cores.

3.3.1 Problema de Quatro Cores

Primeiramente, imaginando que a rede neuronal deve ser capaz de aprender a inverter o efeito da degradação da imagem de entrada, espera-se que seu comportamento não esteja influenciado pelas imagens de treinamento [1]. Portanto, para averiguar essa capacidade da rede, testou-se a restauração de uma décima primeira imagem, que não estava presente no conjunto de treinamento.

Para avaliar a qualidade das imagens restauradas, utilizou-se o índice de similaridade estrutural média (*Mean Structural Similarity* - MSSIM). O MSSIM mede a similaridade entre duas imagens e funciona como uma medida de qualidade de uma imagem em relação a outra, considerando características do sistema visual humano. Essa medida assume valor no intervalo $[0, 1]$, sendo igual a um quando as duas imagens são iguais.

As imagens degradadas são mostradas na Figura 25, obtidas com a PSF gaussiana com diferentes valores de desvio padrão. Além disso, é mostrado o valor do índice MSSIM entre a imagem degradada e a original. Já na Figura 26, observa-se a restauração correspondente a cada degradação, com o respectivo valor de MSSIM, resultado obtido pela rede neuronal.

(a) $\sigma = 0.5$ / 0.99

(b) $\sigma = 1$ / 0.90

(c) $\sigma = 1.5$ / 0.82

(d) $\sigma = 2$ / 0.77

(e) $\sigma = 3$ / 0.68

Figura 25: Imagens degradadas com o σ a esquerda e o MSSIM a direita.

(a) MSSIM 0.99

(b) MSSIM 0.97

(c) MSSIM 0.88

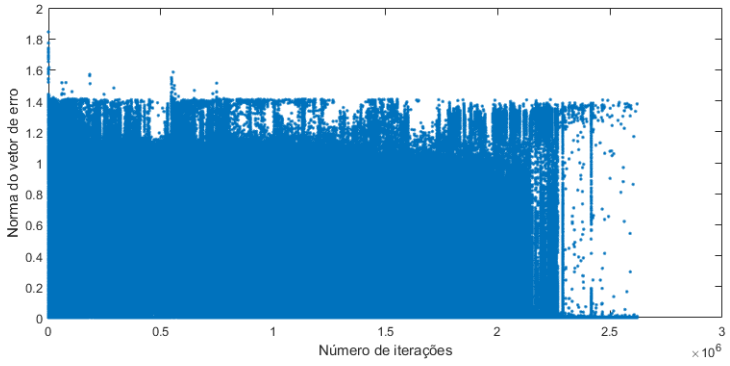
(d) MSSIM 0.82

(e) MSSIM 0.72

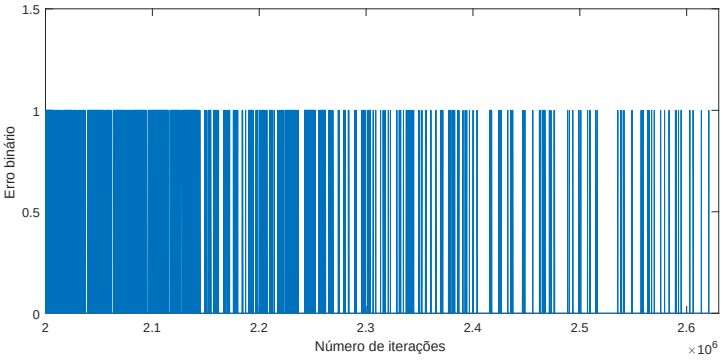
Figura 26: Imagens obtidas após restauração.

Analisando as figuras restauradas, observa-se que, para valores de $\sigma \leq 1$, foi possível restaurar a imagem quase perfeitamente, com MSSIM $\simeq 1$. Por outro lado, para valores de σ elevados, como $\sigma = 3$, a rede não restaurou adequadamente a imagem, obtendo o MSSIM= 0,72 e perdendo parte da informação que a imagem original continha. Assim, para continuar com os testes, foi realizado para $\sigma = 2$ um treinamento com 8 épocas, obtendo MSSIM= 0,86. O problema está no aumento do tempo de treinamento conforme o aumento do número de épocas.

Além da análise das imagens restauradas, outra forma de observar os resultados é averiguar o erro ao longo das iterações. Assim, calculou-se a norma do vetor erro obtido na saída da rede a cada iteração do algoritmo, apresentado no primeiro gráfico. Para simplificar a visualização do erro, é mostrado também um gráfico com os acertos e erros da rede (0 para acerto e 1 para erro, ou seja, binário). Esses dois tipos de gráficos são apresentados nas Figura 27 e 28 para $\sigma = 0.5$ e $\sigma = 2$, respectivamente.



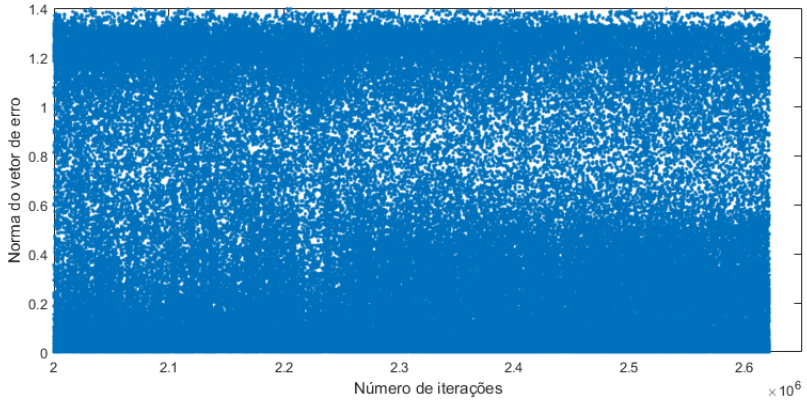
(a) Norma do vetor de erro



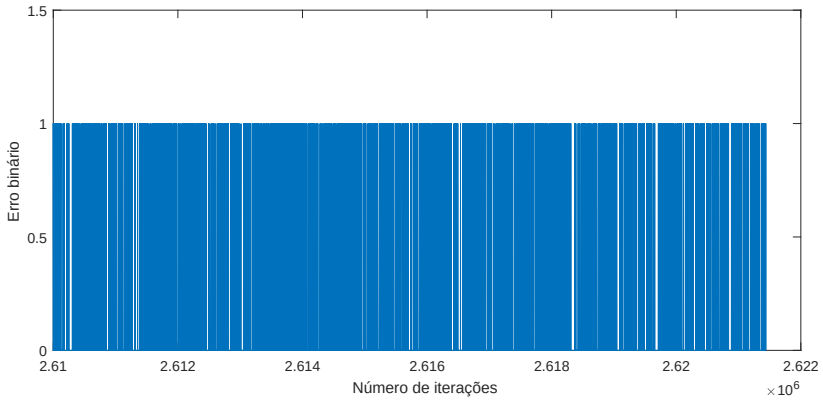
(b) Acertos e erros ao longo das iterações

Figura 27: Gráficos do erro para $\sigma = 0.5$.

Ao observar os erros obtidos para $\sigma = 0.5$, nota-se que a norma do erro é minimizada ao longo das iterações. Assim, quando o treinamento se aproxima do fim, a rede erra menos a estimativa dos pixels da imagem original. Tal fato é visto também com maior facilidade no gráfico de erro binário, onde as barras em 1, que indicam uma cor selecionada errada, se tornam espaçadas conforme as iterações, ou seja, diminuição na quantidade de erros cometidos pela rede neuronal.



(a) Norma do vetor de erro



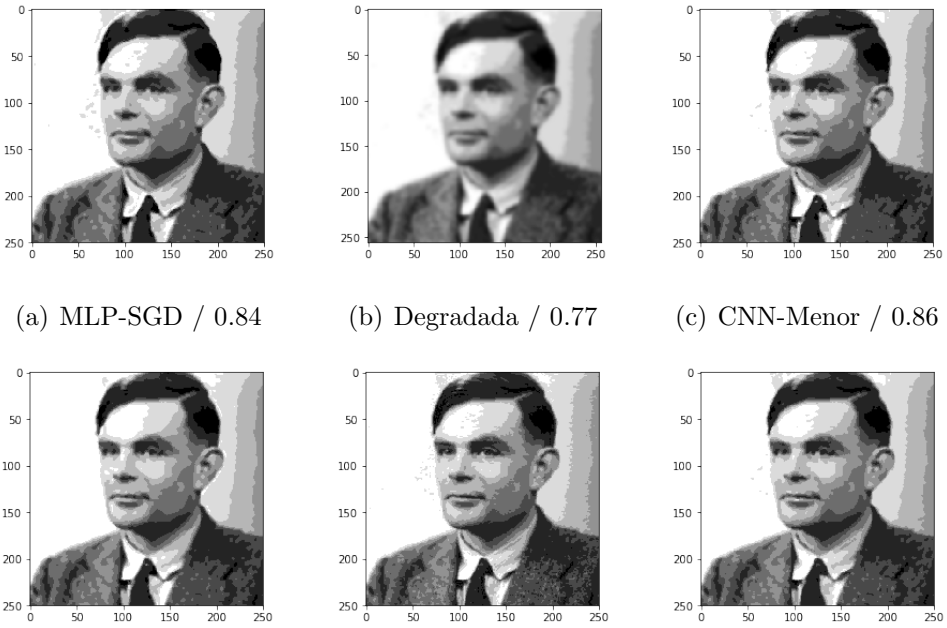
(b) Erro binário

Figura 28: Gráficos do erro para $\sigma = 2$.

Ao contrário dos erros observados anteriormente, considerando agora $\sigma = 2$, uma grande quantidade de erros ainda ocorre mesmo no final das iterações. Isso se deve à perda de informação que a rede sofre ao restaurar imagens com forte degradação. Como comprovação, o gráfico de erro binário apresenta poucas regiões no eixo x em que há uma sequência de acertos.

3.3.2 Problema de Oito Cores

No problema em questão, as redes treinadas foram testadas com uma vigésima imagem que não estava presente no conjunto de treinamento, assim como foi feito para o problema de quatro cores. Além disso, para avaliar o resultado obtido após a restauração das imagens, também foi utilizado o índice MSSIM. Como as quatro redes utilizaram a mesma função custo, a entropia cruzada, os valores da função custo a cada época para a simulação de cada rede são comparados. Os resultados obtidos em 10 épocas para as quatro redes são apresentados a seguir.



(a) MLP-SGD / 0.84 (b) Degradada / 0.77 (c) CNN-Menor / 0.86

(d) MLP-Adam / 0.83 (e) Original (f) CNN-Maior / 0.87

Figura 29: Imagens restauradas em 10 épocas com a respectiva rede na esquerda e o índice MSSIM na direita.

Como foi possível observar, o melhor resultado obtido foi o da rede CNN-Maior, elevando em 0,1 o índice MSSIM, comparando seu resultado com a imagem degradada. Assim, as técnicas convolucionais se apresentaram ligeiramente melhores do que as redes MLP, comparando-se os índices MSSIM. O aumento da quantidade de filtros entre as duas redes CNNs significou uma pequena melhora no índice também. A maior diferença ocorreu entre as redes MLP-Adam e CNN-Maior, sendo de 0,04 entre os índices MSSIM.

É possível observar a diferença nos detalhes das imagens. Como exemplo, observa-se algumas manchas cinzas presentes ao lado esquerdo do rosto na imagem original e no resultado das redes CNN. Já na rede MLP-Adam, o mesmo não é observado.

Outra forma de avaliar o desempenho das redes é através do gráfico de valores da função custo por época, apresentado a seguir para cada rede.

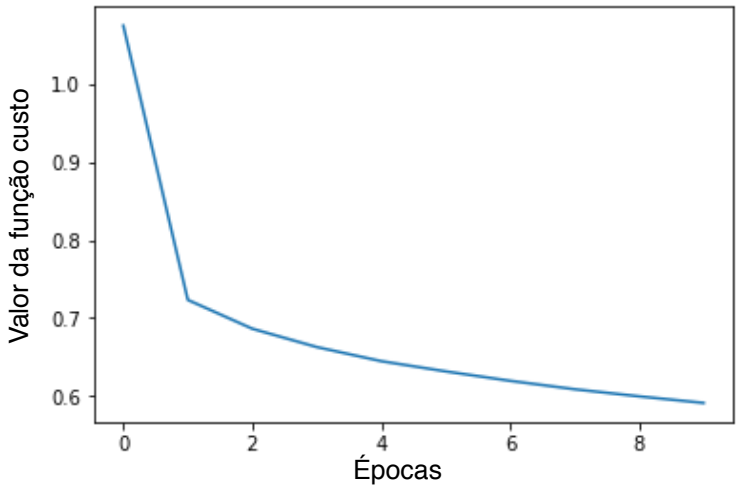


Figura 30: Gráfico da função custo em 10 épocas para a rede MLP-SGD.

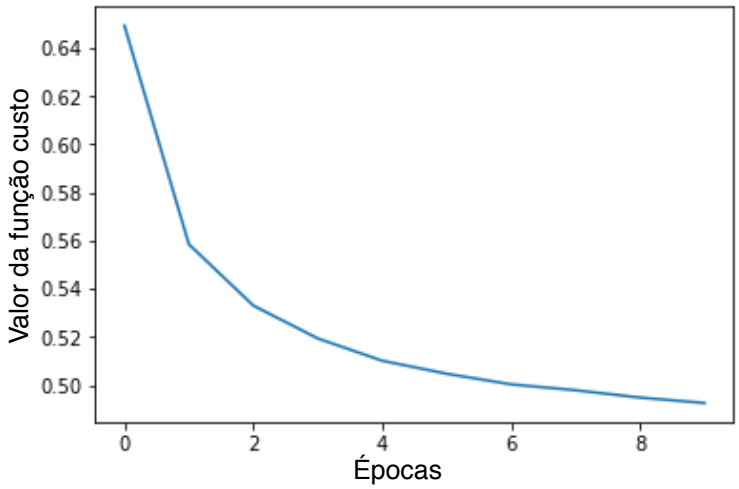


Figura 31: Gráfico da função custo em 10 épocas para a rede MLP-Adam.

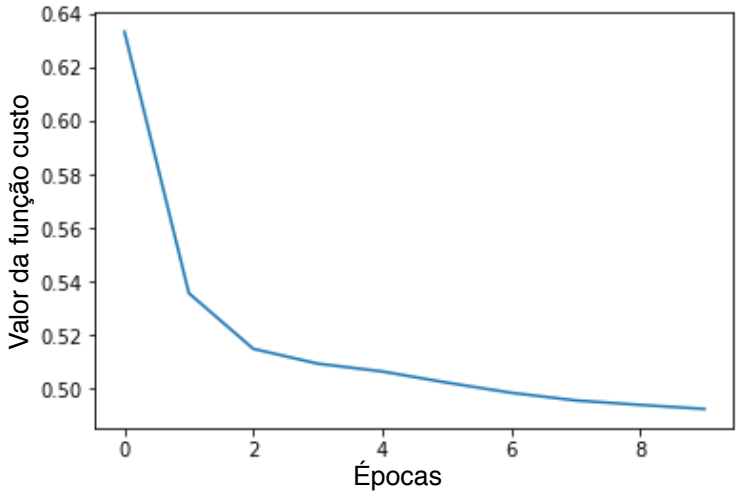


Figura 32: Gráfico da função custo em 10 épocas para a rede CNN-Menor.

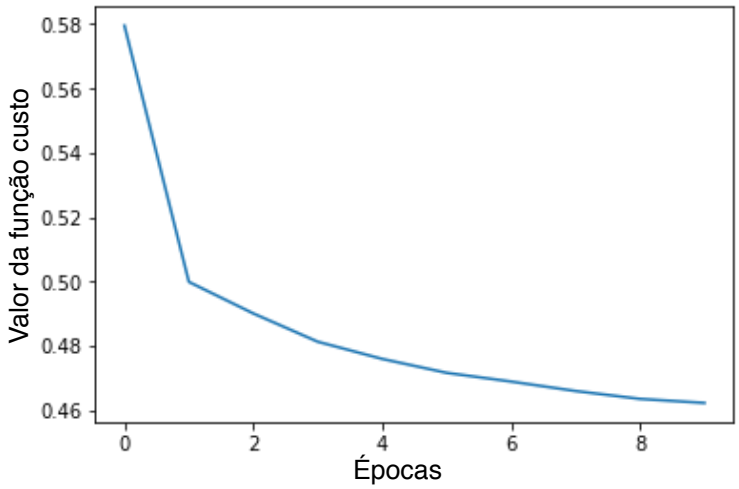
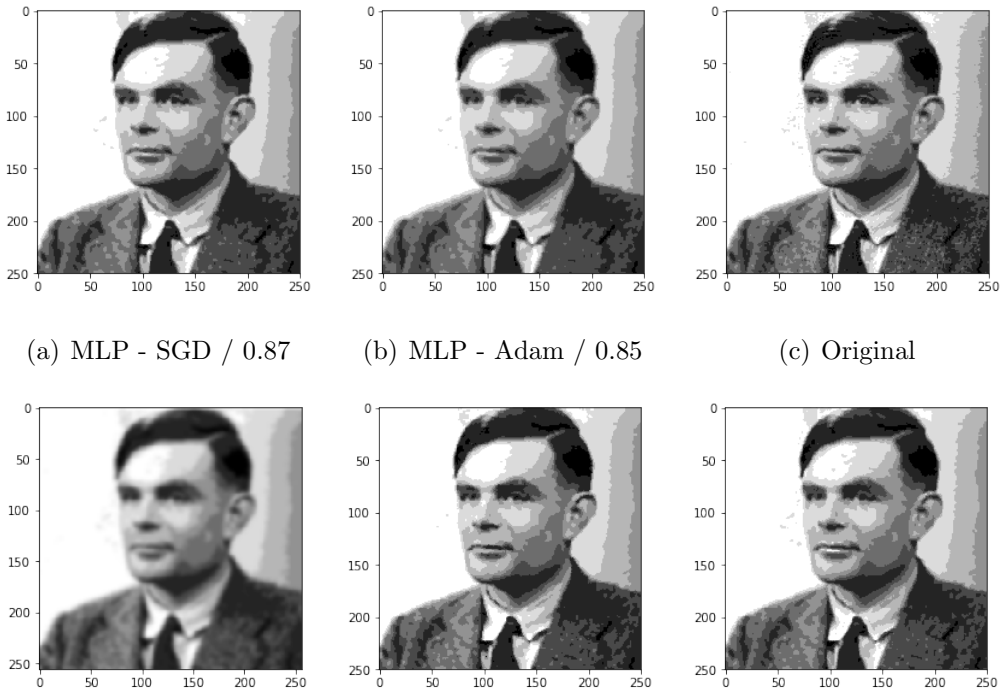


Figura 33: Gráfico da função custo em 10 épocas para a rede CNN-Maior.

A grande diferença observada entre as redes através dos gráficos anteriores está na velocidade de convergência. O tempo médio levado para a simulação de cada época para as redes MLP-SGD, MLP-Adam, CNN-Menor e CNN-Maior foram, respectivamente, de $132s$, $141s$, $161s$ e $189s$. As simulações foram realizadas com o processador Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz 2.40GHz. Assim, a rede CNN-Maior foi a primeira a obter um valor de 0,5 para a função custo, alcançado em apenas uma época. Já

para as outras redes, o mesmo valor para a função custo foi obtido algumas épocas depois, significando um tempo de simulação maior e, consequentemente, aumento do custo computacional.

A seguir, são apresentados os resultados obtidos pelas quatro redes após o treinamento de 100 épocas.



(a) MLP - SGD / 0.87 (b) MLP - Adam / 0.85 (c) Original

(d) Degradada / 0.77 (e) CNN - Menor / 0.86 (f) CNN - Maior / 0.87

Figura 34: Imagens restauradas em 100 épocas com a respectiva rede na esquerda e o índice MSSIM na direita.

Desta vez, a rede MLP-SGD obteve o mesmo índice MSSIM que a rede CNN-Maior, valendo 0,87. Vale ressaltar que a rede MLP-Adam também melhorou o seu resultado do índice MSSIM de 10 para 100 épocas, já a rede CNN-Menor não. Em seguida, são apresentados os gráficos dos valores da função custo para 100 épocas.

Como é possível observar, as redes CNN obtiveram menores valores para a função custo com um menor número de épocas se comparadas com as redes MLP. Os mesmos valores de tempo para cada época obtidos para simulação de 10 épocas valem aqui também. Isso acarreta num menor custo computacional para as redes CNNs, já que a CNN-Maior levou, aproximadamente,

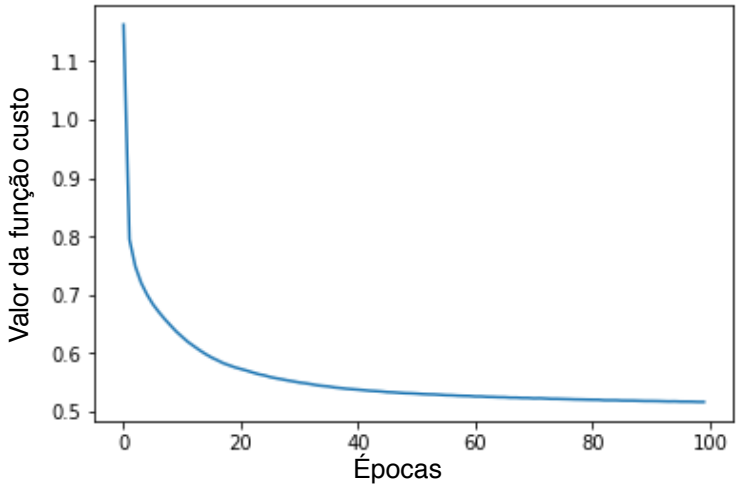


Figura 35: Gráfico da função custo em 100 épocas para a rede MLP - SGD.

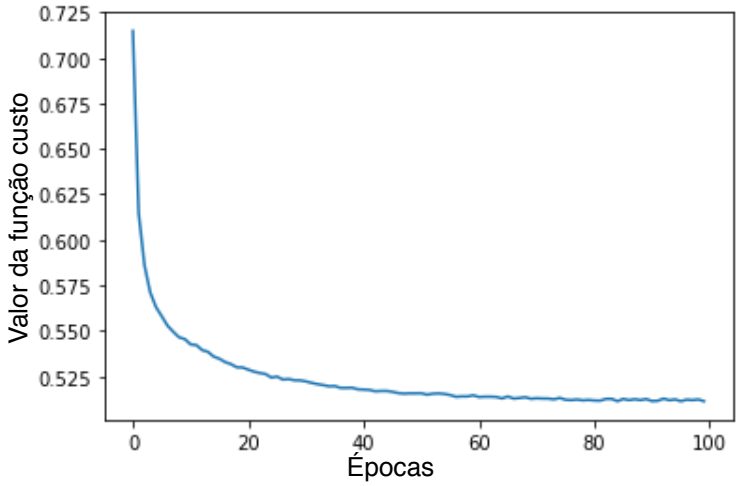


Figura 36: Gráfico da função custo em 100 épocas para a rede MLP - Adam.

10 épocas para alcançar MSSIM 0,87 e a rede MLP - SGD precisou de mais épocas para obter o mesmo resultado. Assim, por mais que essas redes tenham obtido resultados semelhantes, as redes CNN se destacam diante das redes MLP quando se trata de custos computacional para o treinamento de redes neuronais.

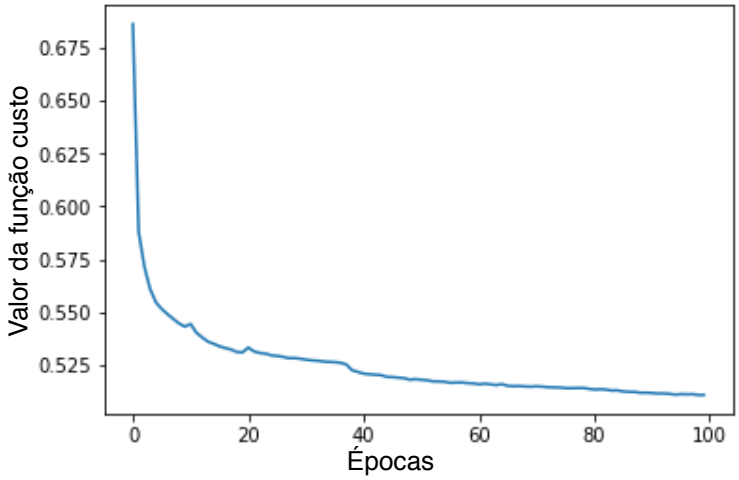


Figura 37: Gráfico da função custo em 100 épocas para a rede CNN - Menor.

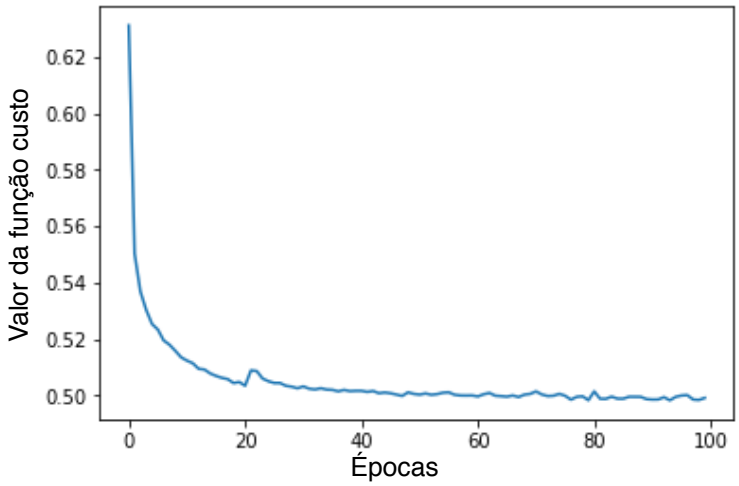


Figura 38: Gráfico da função custo em 100 épocas para a rede CNN - Maior.

4 Conclusão

Após a análise dos resultados obtidos com a restauração de imagens, as simulações com as redes neuronais se demonstraram relevantes, visto que o índice MSSIM das imagens restauradas foram maiores que o das imagens degradadas para todos os problemas enfrentados, melhorando a qualidade da imagem. Em contrapartida, conforme a complexidade dos desafios apresentados aumentou, por exemplo, quando se elevou o número de cores das imagens

trabalhadas, o custo computacional para as simulações também cresceu.

Vale ressaltar também que se constatou a eficiência das redes CNN se comparadas com as redes MLP. Para o problema de oito cores, mesmo quando o índice MSSIM foi igual entre as redes CNN-Maior e MLP-SGD, a rede neuronal convolucional obteve o resultado em um tempo de simulação consideravelmente menor.

Portanto, comparando o trabalho desenvolvido neste relatório com o cronograma proposto para a iniciação científica, observa-se que o estudo de redes neuronais e a aplicação de redes profundas para a restauração de imagens degradadas por uma PSF específica foram atividades bem sucedidas. A implementação de uma solução cega, ou seja, PSF desconhecida, não foi abordada, visto a complexidade do problema [4, 11] e o curto período de iniciação científica.

5 Anexos

```
import keras
from keras.models import Sequential
from keras.layers import Flatten, Conv2D, Dense, Activation
from keras import optimizers

labels = np.rint(labels)
t_labels = np.rint(t_labels)
train_labels = keras.utils.to_categorical(labels, num_classes=max_color + 1)
train_data = data
test_labels = keras.utils.to_categorical(t_labels, num_classes=max_color + 1)
test_data = t_data

model = Sequential()

model.add(Conv2D(8, (3, 3), input_shape=(7, 7, 1)))
model.add(Activation('relu'))
model.add(Conv2D(8, (3, 3)))
model.add(Activation('relu'))
model.add(Conv2D(8, (3, 3)))
model.add(Activation('relu'))
model.add(Flatten())
model.add(Dense(8))
model.add(Activation('softmax'))

opt = optimizers.Adam(lr=0.001,
                      beta_1=0.9,
                      beta_2=0.999,
                      epsilon=None,
                      decay=0.0,
                      amsgrad=False)

model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['categorical_accuracy'])

model.summary()

history = model.fit(train_data, train_labels, epochs=10, batch_size=32)

score = model.evaluate(x=test_data, y=test_labels, batch_size=1)

test_output = model.predict(test_data, batch_size=1)
```

Figura 39: Trecho do código da rede CNN-Maior utilizando o Keras.

Referências

[1] M. Egmont-Petersen, D. de Ridder, and H. Handels, "Image processing with neural networks – a review," *Pattern Recognition*, vol. 35, no. 10, pp. 2279–2301, 2002.

[2] S. S. Haykin, *Neural networks and learning machines*, vol. 3, Pearson Upper Saddle River, NJ, USA:, 2009.

[3] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[4] A. Lucas, M. Iliadis, R. Molina, and A. K. Katsaggelos, "Using deep neural networks for inverse problems in imaging: Beyond analytical methods," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 20–36, Jan 2018.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

[6] D. Kundur and D. Hatzinakos, "Blind image deconvolution," *IEEE Signal Processing Magazine*, vol. 13, pp. 43–64, May 1996.

[7] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[8] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Prentice Hall, NJ, 3rd edition, 2008.

[9] R. A. Hummel, B. Kimia, and S. W. Zucker, "Deblurring gaussian blur," *Computer Vision, Graphics, and Image Processing*, vol. 38, no. 1, pp. 66–80, 1987.

[10] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[11] Li Xu, Jimmy S. J. Ren, Ce Liu, and Jiaya Jia, "Deep convolutional neural network for image deconvolution," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, Cambridge, MA, USA, 2014, NIPS'14, pp. 1790–1798, MIT Press.