Remoção de *Motion Blur* Horizontal por Classificação do *Kernel* no Domínio da Frequência usando Rede Neural Convolucional

Tafnes Silva Barbosa[1] e Rodrigo Carvalho de Paulo[2]

***Resumo*— Com a crescente utilização de visão computacional na ciência e na engenharia, remover distorções como *motion blur* tem se mostrado um desafio a ser superado. Técnicas do estado-da-arte têm utilizado aprendizado de máquina profundo no domínio do espaço para endereçar esse problema, mas antes do desenvolvimento dessa área o domínio da frequência era bastante empregado. Este trabalho tem por propósito construir uma rede neural convolucional para remoção de *motion blur* horizontal treinada para identificar o *kernel*, no domínio da frequência, por meio de uma transformada de Fourier discreta – DFT da imagem distorcida. Para tanto, foi treinada uma adaptação da rede neural YOLOv1 no domínio da frequência a partir de um *dataset* de imagens com *motion blur* horizontal em números ímpares de *kernel* `[3, 5, 7, 9, 11, 13]`. A rede implementada obteve uma acurácia de 99,85% para um *dataset* de teste de 15% de um total de 31723 imagens distribuídas em 70% para treinamento e 15% para validação. Além disso, a rede foi eficaz na recuperação da imagem mesmo com o ruído inserido por sua compressão. Assim, este resultado contribui para área, pois implementa uma abordagem diferente da utilizada pelo estado-da-arte e permite a exploração da técnica para outros problemas relacionados, como remoção *motion blur* não uniforme e determinação de fluxo de movimento no domínio da frequência.**

***Palavras-chave*— Motion Blur, Rede Neural Convolucional, Kernel, DFT, Domínio da Frequência**

I. INTRODUÇÃO TEÓRICA

A. Contexto

Com o avanço do uso de visão computacional em diversas áreas da ciência e da engenharia, como na utilização em carros autônomos e robôs móveis, as distorções presentes em imagens podem ser um obstáculo para a aplicação desse tipo de tecnologia. Mais especificamente, a presença de *motion blur* em imagens é um tipo de distorção muito conhecida e que pode impossibilitar diversas tarefas que dependam da nitidez da imagem, como o reconhecimento de texto em uma imagem. Ela ocorre de forma local quando algum objeto está em movimento durante a fotografia ou de forma global quando é a câmera fotográfica que se move.

B. Descrição Matemática do Problema

Apesar de sua origem estar ligada ao momento de criação da imagem, esta distorção pode ser simulada ao se aplicar filtros convolucionais, que retornam a média dos *pixels* numa certa região para uma dada direção de movimento. A equação a seguir mostra como essa função é aplicada:

$$g(x,y) = h(x,y) * f(x,y), \tag{1}$$

em que $f(x,y)$ é a imagem de tamanho $M \times N$, $h(x,y)$ é o filtro aplicado (também chamado de *kernel*), $g(x,y)$ é a imagem com *motion blur* e $*$ representa a operação de convolução. Esse *kernel* pode ter vários tamanhos e costuma ter tamanho ímpar. No caso deste estudo, utilizou-se *motion blur* horizontal que é dado pela seguinte equação:

$$h(x,y) = \begin{cases} \frac{1}{2p+1}, & \text{se } x = p \\ 0 & c.c. \end{cases}, \tag{2}$$

em que o tamanho do kernel é dado por $(2p+1 \times 2p+1)$ e os índices das linhas e colunas vão de 0 a $2p$. Por exemplo, um filtro de tamanho 3 é:

$$h(x,y) = \frac{1}{3} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Pelo fato de somente a linha central do *kernel* ser diferente de zero, é possível facilitar a visualização da convolução através da seguinte forma:

$$g(x,y) = \frac{1}{2p+1} \cdot \sum_{t=-p}^{p} f(x,y-t), \tag{3}$$

em que o *kernel* tem tamanho $2p+1$ e é realizado um *zero padding* na imagem para que, nas suas bordas, não haja problema com valores negativos de x e y. Essa mudança foi realizada com o intuito de tornar mais fáceis as operações matemáticas necessárias a serem aplicadas posteriormente.

A Figura 1a mostra uma imagem do *dataset* usado nesse estudo e a Figura 1b mostra a mesma imagem com *motion blur* simulado a partir de um *kernel* de tamanho 9.

(a) Sem *motion blur*.

(b) Com *motion blur* simulado a partir de um *kernel* de tamanho 9.

Fig. 1: Imagem retirada do *dataset* usado no treinamento.

No entanto, as imagens podem ser estudadas não somente pelo domínio espacial, mas também pelo domínio

[1]Tafnes Silva Barbosa trabalha no Instituto de Pesquisas e Ensaios em Voo, Brasil, `tafnessilvabbarbosa@gmail.com`

[2]Rodrigo Carvalho de Paulo trabalha no Instituto de Pesquisas e Ensaios em Voo, Brasil `rodrigocdpaulo@gmail.com`

da frequência. E este domínio pode ser acessado através da Transformada de Fourier Discreta (DFT, do inglês: *Discrete Fourier Transform*), dada na equação a seguir:

$$F(u,v) = DFT\{f(x,y)\} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-2\pi i\left(\frac{ux}{M}+\frac{vy}{N}\right)}, \quad (4)$$

em que $f(x,y)$ é a imagem e i é o número complexo $i = \sqrt{-1}$.

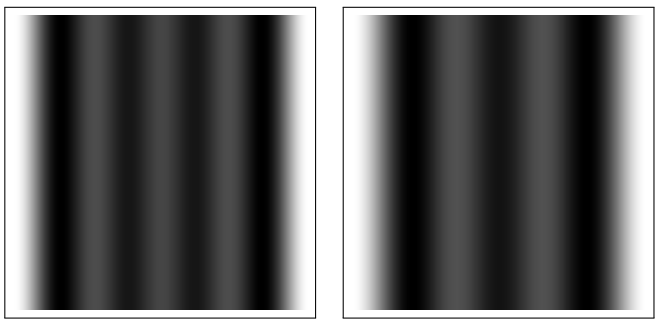
A partir da transformada de Fourier, pode-se obter a imagem distorcida no domínio da frequência a partir da equação a seguir:

$$\begin{aligned} G(u,v) &= DFT\{g(x,y)\} \\ &= DFT\left\{\frac{1}{2p+1}\cdot\sum_{t=-p}^{p} f(x,y-t)\right\} \\ &= \frac{1}{2p+1}\cdot\sum_{t=-p}^{p} DFT\{f(x,y-t)\} \\ &= \frac{1}{2p+1}\cdot\sum_{t=-p}^{p} F(u,v)\cdot e^{\frac{-2\pi ivt}{N}} \\ &= \frac{F(u,v)}{2p+1}\cdot\sum_{t=-p}^{p} e^{\frac{-2\pi ivt}{N}} \\ &= \frac{F(u,v)}{2p+1}\cdot\left[1+\sum_{t=1}^{p} 2\cos\left(\frac{2\pi vt}{N}\right)\right] \\ &= F(u,v)H(u,v) \end{aligned} \quad (5)$$

Enquanto um computador pode aplicar a DFT às imagens de forma muito eficiente sem a necessidade de saber a equação matemática, percebeu-se que o conhecimento dela explica a utilização das redes neurais para classificação da distorção. Usando transformações trigonométricas pode-se reduzir a função $H(u,v)$ a

$$H(u,v) = \frac{1}{2p+1}\cdot\left[1+\frac{2\sin\left(\frac{p\pi v}{N}\right)\cos\left(\frac{(p+1)\pi v}{N}\right)}{\sin\left(\frac{\pi v}{N}\right)}\right], \quad (6)$$

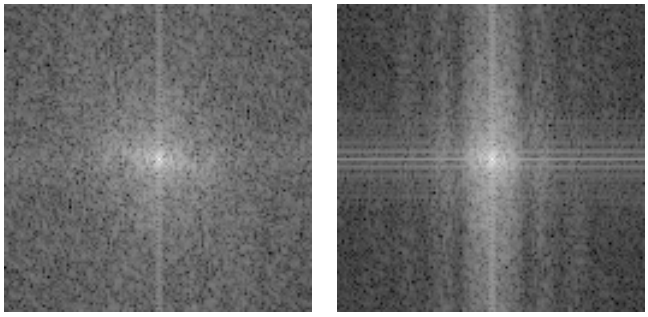
que é uma função limitada e com ondas na sua formação. A Figura 2 mostra a função $H(u,v)$ para $p = 4$ e $p = 3$, ou seja, para o *kernels* de tamanho 9 e 7, respectivamente, mas como imagens de tamanho $M \times N$, do mesmo tamanho da imagem original (Figura 1a).



(a) *Kernel* de tamanho 9. (b) *Kernel* de tamanho 7.

Fig. 2: Função $H(u,v)$ para *kernels* de tamanhos 9 e 7.

Percebe-se que para cada *kernel* o formato da função $H(u,v)$ é diferente e como essa função é multiplicada à DFT da imagem original, no domínio da frequência, pode-se ver diferentes padrões de oscilações na DFT das imagens com *motion blur*. A Figura 3 mostra as faixas verticais, que são as oscilações, que aparecem na DFT da imagem com distorção.



(a) Imagem original. (b) Imagem com *motion blur*.

Fig. 3: FFT das imagens original e com *motion blur*, na qual pode-se ver oscilações (faixas verticais mais escuras com espaçamentos entre si adicionadas).

Com base nisso, neste artigo nós apresentamos a construção de uma rede neural convolucional (CNN, do inglês: *Convolutional Neural Network*) para classificar o *motion blur* horizontal concernente a uma imagem no domínio da frequência a partir de sua DFT.

C. Trabalhos Relacionados

Estudos anteriores ao desenvolvimento do aprendizado de máquina profundo utilizavam o domínio da frequência para remoção de *montion blur* com diferentes técnicas como o *Point Spread Function* (PSF) [1][2]. Recentemente, alguns trabalhos [3][4] propuseram o uso do domínio da frequência sem aprendizado de máquina. Porém, após o advento do aprendizado de máquina profundo inúmeros trabalhos utilizam redes neurais profundas para tratar o problema no domínio do espaço, considerado o estado-da-arte [5][6][7]. Contudo, foram encontrados poucos trabalhos que utilizam a combinação do domínio da frequência e aprendizado de máquina profundo como o artigo [8], que utiliza transformada *wavelet*, sendo desconhecido trabalhos que utilizam a transforma de *Fourier* com rede neural profunda para tratar o problema.

Portanto, a contribuição desse artigo encontra-se na proposta da utilização da técnica, aprendizado de máquina profundo, como apoio para a remoção de *motion blur* horizontal de imagens utilizando uma rede neural convolucional treinada no domínio da frequência.

II. Metodologia

A. Classificação de Motion Blur Usando Rede Neural Convolucional

A ideia inicial para este projeto foi adaptar a rede U-net [9], usada para segmentação em imagens, para corrigir diretamente as distorções devido a *motion blur*. Porém, ao se implementar a rede e realizar algumas tentativas de treinamento não se percebeu um resultado bom. Tinha se usado a métrica *cosine similarity* e *mean squared error*, pois

a tentativa era se obter a imagem mais próxima da original a partir da imagem com distorção, mas as métricas mostravam que a rede não estava atingindo a capacidade esperada. Por isso, decidiu-se mudar a abordagem. Então, foi montada uma CNN baseada na YOLOv1 [10] para classificação das distorções, para se recuperar a imagem com técnicas mais específicas, como será abordado posteriormente.

Ao contrário da YOLO, nossa rede usa convoluções sem *zero padding* o que diminui o tamanho de cada camada após as convoluções.

A entrada da rede é uma imagem de tamanho (128x128) com um único canal de cor, pois é a DFT da imagem original. A rede contém três blocos de convoluções, onde cada bloco termina com um *MaxPooling*, e duas camadas totalmente conectadas (FC, do inglês: *Fully Connected*) no final da rede. Todas as camadas, com exceção da última, tem a função Leaky ReLU como ativação com $\alpha_{\text{Leaky ReLU}} = 0.1$. A Tabela I mostra a arquitetura da CNN usada.

TABELA I: Arquitetura da rede neural convolucional usada.

Tipo	*Size*	*Stride*	Ativação	Formato Saída
Entrada	–	–	–	128x128x1
CONV	3x3	1x1	Leaky ReLU	126x126x192
POOL	2x2	2x2	–	63x63x192
CONV	1x1	1x1	Leaky ReLU	63x63x128
CONV	3x3	1x1	Leaky ReLU	61x61x256
CONV	1x1	1x1	Leaky ReLU	61x61x256
CONV	3x3	1x1	Leaky ReLU	59x59x512
POOL	2x2	2x2	–	29x29x512
CONV	1x1	1x1	Leaky ReLU	29x29x256
CONV	3x3	1x1	Leaky ReLU	27x27x512
CONV	1x1	1x1	Leaky ReLU	27x27x256
CONV	3x3	1x1	Leaky ReLU	25x25x512
CONV	1x1	1x1	Leaky ReLU	25x25x256
CONV	3x3	1x1	Leaky ReLU	23x23x512
CONV	1x1	1x1	Leaky ReLU	23x23x256
CONV	3x3	1x1	Leaky ReLU	21x21x512
CONV	1x1	1x1	Leaky ReLU	21x21x512
CONV	3x3	1x1	Leaky ReLU	19x19x1024
POOL	2x2	2x2	–	9x9x1024
FLATTEN	–	–	–	82944
FC	–	–	Leaky ReLU	24
FC	–	–	Softmax	6
Número de parâmetros, todos treináveis: 13,786,798.				

Para cada camada da rede, também foi provisionado para usos futuros a regularização do tipo L2 dos *kernels* de convolução. No caso deste projeto, usou-se $\lambda_{reg} = 0$.

A saída da rede tem 6 neurônios, um para cada classe do *dataset*. Como se usou a codificação *one-hot*, usou-se a função de ativação Softmax na última camada. A penúltima camada tem 4 vezes o número de classes. Esse valor foi definido após *benchmark* com a YOLOv1 que possui 1000 classes e sua penúltima camada tem 4096 neurônios.

B. Recuperação da imagem com motion blur

Como falado, o *motion blur* horizontal pode ser simulado conforme a equação (3). Esta equação também pode ser vista da forma de multiplicação de matrizes:

$$g = (Hf^T)^T, \tag{7}$$

em que $g_{M\times N}$ é a imagem com *motion blur*, $f_{M\times N+2p}$ é a imagem original com *zero padding* de forma a permitir a multiplicação com a matriz $H_{N\times N+2p}$ que é:

$$H(x,y) = \begin{cases} \frac{1}{2p+1}, & \text{se } x \le y \le x+2p \\ 0, & c.c. \end{cases}, \tag{8}$$

para $0 \le x \le M-1$ e $0 \le y \le N-1$, em que o tamanho do *kernel* é igual a $2p+1$. Por exemplo, para o *kernel* de tamanho 3, tem-se:

$$H = \frac{1}{3} \cdot \begin{bmatrix} 1 & 1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 1 & 1 & \cdots & 0 \\ 0 & 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}_{(N\times N+2)} \tag{9}$$

O *zero padding* de f é feito da seguinte forma: coloca-se p colunas de zeros à esquerda e p colunas de zeros à direita.

Para recuperar a imagem f, basta resolver o sistema de equações dado pela equação (7). No entanto, esse sistema tem infinitas soluções. Uma solução é tomar a solução que minimiza o erro quadrático médio ao fazer:

$$g^T = Hf^T \Rightarrow H^T g^T = H^T H f^T \Rightarrow \left(H^T H\right)^{-1} H^T g^T = f^T$$

Logo, tem-se que

$$f = \left(H^+ g^T\right)^T, \tag{10}$$

em que $H^+ = \left(H^T H\right)^{-1} H^T$ é também chamada de matriz pseudo-inversa de H ou matriz Moore-Penrose inversa de H. Em python, essa matriz pode ser encontrada através da biblioteca NumPy:

```
Hplus = np.linalg.pinv(H)
```

Pode-se pensar que uma abordagem mais correta removeria p colunas da esquerda H e p colunas da sua direita, dado que a imagem original não tem os zeros de preenchimento. Ainda assim, o sistema de equações não pode ser resolvido pelo fato de a matriz H obtida ser singular. Ao se aplicar a pseudo-inversa, o resultado obtido não é tão bom quanto o da primeira forma (sem remover as colunas devido ao *zero padding*). A Figura 4 mostra uma comparação entre as duas formas. Por isso, nesse trabalho, usou-se o primeiro método para a recuperação das imagens.

(a) Com primeiro método. (b) Removendo as colunas de H.

Fig. 4: Imagens recuperadas com os métodos mencionados acima.

C. Recuperação de Imagem do Dataset

Como será explicado mais adiante, o *dataset* foi criado aplicando-se convolução com *kernels* de tamanhos predefinidos e as imagens foram salvas no formato JPG. Este formato faz com que haja perda de informação.

Percebeu-se que a diferença entre os *pixels* de uma imagem antes de salvar em JPG e depois é semelhante à adição de ruído gaussiano à imagem. A Figura 5 mostra um histograma do ruído (diferença entre as imagens antes e depois de salvar em JPG) em azul e a curva gaussiana obtida partir da média e desvio padrão dos ruídos assumindo que a integral da curva gaussiana é igual à integral do histograma (a integral do histograma não considera valores fora do plotado, por isso a verdadeira curva gaussiana derivada pode ser um pouco diferente da apresentada).

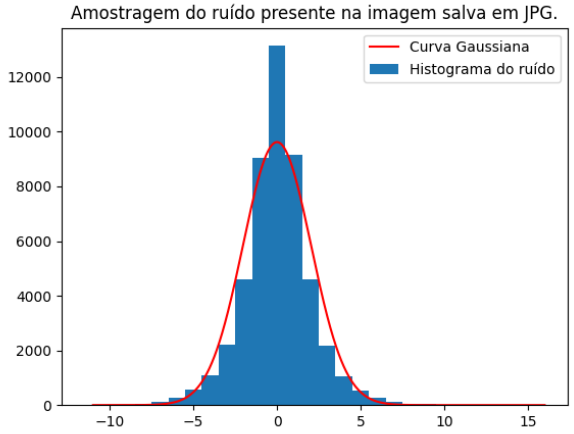


Fig. 5: Diferença entre imagem antes e depois de salvar em JPG se comportando como um ruído gaussiano.

A adição de ruído gaussiano torna o modelo mais próximo da realidade dado que fotografias sempre apresentam ruído gaussiano. Com isso, a imagem com *motion blur* pode ser dada pela seguinte fórmula, que é uma adaptação da equação (3):

$$g(x,y) = h(x,y) * f(x,y) + n(x,y), \tag{11}$$

em que $n(x,y)$ é o ruído gaussiano.

III. EXPLICAÇÃO DO CÓDIGO IMPLEMENTADO

A. Criação do Dataset

O *dataset* foi criado a partir de um conjunto de 31783 imagens disponíveis no Kaggle, as quais pertencem ao *dataset* Flickr30k [11]. Segundo a descrição do site, este *dataset* tornou-se um padrão de referência para a descrição de imagens baseadas em frases. No entanto, usou-se somente as imagens deste *dataset* por causa da grande quantidade de imagens e dos mais diversos tipos.

Como as imagens tinham vários tamanhos, todas elas foram reajustadas para terem o tamanho de entrada da nossa rede (128x128) usando o método `nearest` da função `resize` pertencente à livraria `tensorflow.image`. Com isso, várias das imagens perderam qualidade como se vê na Figura 1a, que parece bastante pixelizada.

Após o reajuste de tamanho, foi aplicado *motion blur* horizontal às imagens através da função `filter2D` da biblioteca `cv2` da OpenCV, que realiza a convolução de um *kernel* fornecido com a imagem. Os *kernels* usados estão descritos na equação (2). Os tamanhos dos *kernels* das distorções aplicadas foram escolhidas de forma aleatória e uniforme dentre 6 tamanhos diferentes: `[3, 5, 7, 9, 11, 13]`. Após isso, salvou-se a imagem no formato JPG inserindo o tamanho do *kernel* usado no nome da imagem. Por exemplo, se a imagem se chamava `image` e a ela foi aplicado um filtro de tamanho 5, ela foi salva com o nome `5_image_blurred`.

Para o treinamento, avaliação e teste da CNN foi criada uma classe com herança da classe `keras.utils.Sequence`, que produz os *batches* a partir do método `__getitem__`. Este método gera um *batch* (x,y), onde x é uma matriz com o logaritmo do módulo das DFTs das imagens e y é uma matriz com as codificações *one-hot* das imagens. Cada imagem em x é normalizada para ter seus valores entre 0 e 1. O formato de x é `(batch_size, img_size[0], img_size[1], num_channels)` e o de y é `(batch_size, num_classes)`.

Esta mesma classe realiza o *shuffle* do *dataset* ao final de cada época através do método `on_epoch_end` para o treinamento estocástico da rede. E por último, esta classe também tem um método `__len__` que retorna o número de *batches* em uma época.

O *dataset* de 31783 imagens foi dividido em 3:

- Treinamento: com 22,272 imagens ($\approx 70\%$);
- Validação: com 4,756 imagens ($\approx 15\%$);
- Teste: com 4,755 imagens ($\approx 15\%$).

B. Rede Neural Convolucional

A rede foi implementada como um objeto da classe `tensorflow.keras.models.Sequential` conforme a Tabela I.

Os *kernels* de convolução da rede foram inicializados usando o método `he_normal`, que consiste de amostrar os pesos de uma camada normalmente com variância $\sigma^2 = \frac{2}{n_{in}}$, em que n_{in} é o número de pesos da camada em si.

Para o treinamento da rede foi usado um `batch_size` de 16 e 30 épocas. Foi usado um *schedule* para o *learning rate*, onde o valor inicial foi de $\alpha_0 = 1 \cdot 10^{-7}$ e caía pela metade sempre que epoch $\%$ $10 = 0$ (o resto da divisão por 10 é igual a 0). Foi usado o otimizador Adam. Como perda foi usada a *categorical cross-entropy*, dado que a saída da rede é codificada pela codificação *one-hot*. E como métrica, foi usada a `categorical_accuracy`.

Como a rede tem muitos parâmetros, foi necessário usar GPUs no seu treinamento e para isso usaram-se as disponibilizadas pelo Google Colab. Mesmo assim, o treinamento levou tempo considerável, dado que cada época demorava cerca de 5 minutos (o que é muito melhor sem GPU, onde o tempo mudava para cerca de 9 horas por época).

No decorrer da implementação, o Google Colab cortava as GPUs, por isso, parte do código salvava os pesos da rede da última melhor época, onde o critério de melhor foi a menor perda do *dataset* de validação. Os pesos foram salvos no *drive* devido à facilidade de acesso. E o treinamento podia continuar a partir da última melhor época mesmo em outra sessão.

O código também salva o histórico do treinamento com as taxas de aprendizado, perdas e acurácias tanto do *dataset* de treinamento como de validação a cada época para análise posterior.

IV. RESULTADOS E DISCUSSÕES

Tendo em vista que a identificação do *kernel* é fundamental para a remoção do *motion blur* e considerando o *dataset* dividido em 70% treinamento, 15% validação e 15% teste, a rede neural convolucional deste trabalho obteve acurácia convergente para próximo de 100%, após a realização de 30 épocas. As Figuras 6 e 7 mostram o histórico de treinamento da rede e a sua evolução para os *datasets* de treinamento e validação.

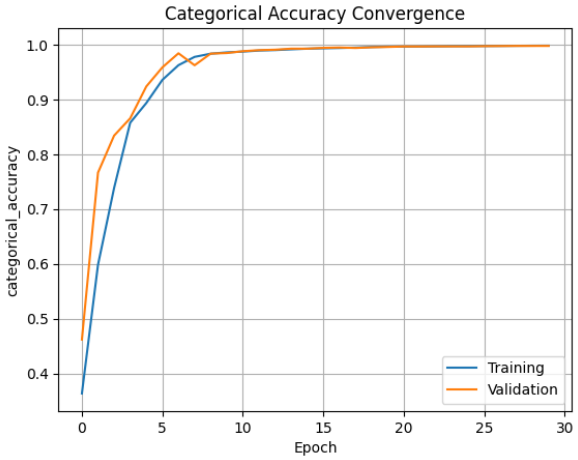


Fig. 6: Gráfico da convergência da acurácia da rede neural convolucional utilizada para o *dataset* de treinamento e validação.

Conforme pode-se ver na Tabela II, que apresenta a performance final da rede após as 30 épocas de treinamento, a acurácia dos *datasets* de treinamento e validação estão bem próximas evidenciando a ausência de *overffiting* da rede. A acurácia do *dataset* de teste próxima à dos outros mostra também que a rede não se ajustou excessivamente ao conjunto de imagens de treinamento+validação por meio da escolha dos hiperparâmetros. Pode-se dizer que a rede está funcionando para diferentes tipos de imagens, dado que o *dataset* de teste nunca é "visto" durante o treinamento.

A rede treinada também exclui a presença de *avoidable bias* dado que a acurácia do *dataset* de treinamento é bem próxima dos 100%. E como já foi visto que não há *overfitting*, tem-se que o modelo não apresenta alta variância. Baixa variância e baixo *bias* confirmam a alta performance do modelo.

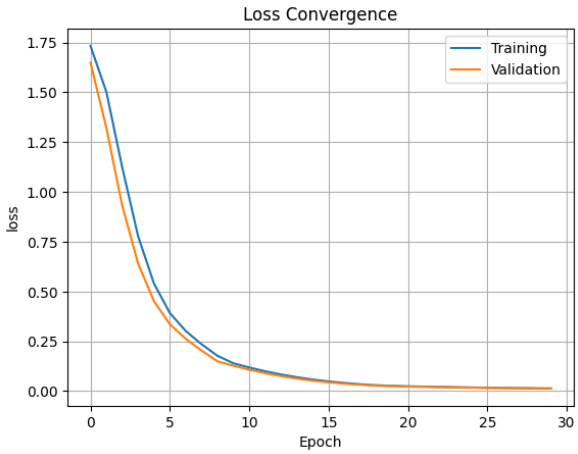


Fig. 7: Gráfico da convergência da função custo da rede neural convolucional utilizada para o *dataset* de treinamento e validação.

TABELA II: Performance da rede neural convolucional treinada.

Dataset	*Categorical accuracy*	*Loss*
70% Treinamento	0,998159	0.013626
15% Validação	0,997685	0.013090
15% Teste	0.998527	0.013300

Por fim, foi então realizada a recuperação da imagem, isto é, a remoção a do *motion blur* horizontal para verificar a efetividade da rede neural de forma visual. É possível observar que a figura obtida não possui mais a distorção causada pelo *motion blur* sendo bastante semelhante a imagem original.

Ao se aplicar o método de recuperação à imagem salva em JPG, ou seja, à imagem com *motion blur* e com ruído gaussiano, obteve-se um resultado mais ruidoso, mas com considerável recuperação. A Figura 8 mostra uma comparação entre as imagens recuperadas antes de salvar em JPG e depois.

(a) Antes de salvar em JPG. (b) Depois de salvo em JPG.

Fig. 8: Imagens recuperadas.

O resultado da Figura 8 mostra como o método de recuperação mencionado pode ser utilizado na recuperação de imagens com *motion blur* após a rede neural convolucional classificar o tipo de distorção concernente à imagem.

Portanto, os resultados obtidos demonstram a eficácia

da abordagem do domínio da frequência para estudos de classificação e remoção de *motion blur* horizontal como também na vertical, que é análogo ao primeiro.

Com base nisso, um dos trabalhos futuros a serem realizados é a utilização da rede para avaliar distorções de *motion blur* não uniformes, dado que as distorções deste estudo são a mesma para toda a imagem. Há de se notar que como a rede neural utilizada é uma adaptação da YOLOv1, que foi desenvolvida para detecção de objetos, consideramos ser possível classificar, para diferentes regiões da imagem, mais de um determinado *kernel*. Assim, para se recuperar a imagem com distorção *motion blur* não uniforme, seria removido o *motion blur* relativo a um *kernel* classificado para cada região da imagem identificada com distorção. Outra abordagem seria utilizar a rede neural desenvolvida para identificar, no domínio da frequência, *motion flow* e a partir disso remover o *motion blur*. Evidentemente, a eficácia da rede neural para imagens reais com *motion blur* é uma etapa importante para validação da rede neural convolucional em trabalhos futuros.

V. CONCLUSÃO

A rede neural convolucional desse trabalho obteve uma acurácia de 99,85% para um *dataset* de teste 15%. Consequentemente, foi possível recuperar uma imagem com *motion blur* horizontal uniforme, mesmo com perda de qualidade devido à compressão. Portanto, esse trabalho foi eficaz em classificar o *motion blur* horizontal com uma rede neural convolucional derivada da YOLOv1 no espectro da frequência utilizando uma DFT. Por fim, esse trabalho utilizou uma técnica de aprendizado de máquina profundo, estado-da-arte para remoção de *motion blur*, com a contribuição de classificar o *kernel* no domínio da frequência. Isso possibilita que trabalhos futuros utilizem essa mesma abordagem para remoção de *motion blur* não uniforme e determinação de *motion flow*.

APÊNDICE

O notebook com a implementação da rede neural convolucional e com o código de recuperação das imagens com *motion blur*, como também o código implementado para gerar o *dataset* com as imagens distorcidas estão disponíveis no repositório do GitHub: `https://github.com/TafnesBarbosa/Remocao-de-Motion-Blur-com-CNN`

Para acessar as imagens do *dataset*, basta rodar o notebook, o qual baixa as imagens de um arquivo zipado no *drive*. Caso queira criar um novo *dataset*, basta usar o arquivo `generate_dataset.py`, colocar a pasta com as imagens distorcidas no *drive* e mudar o id usado para baixar as imagens no respectivo local do notebook.

AGRADECIMENTOS

Agradecemos a Deus por proporcionar a capacidade de estudar, pesquisar e se desenvolver academicamente. Agradecemos ao professor Marcos Máximo pelas aulas e elucidações quanto ao vasto mundo da inteligência artificial. E agradecemos ao Instituto de Pesquisas e Ensaios em Voo - IPEV por nos conceder tempo necessário para nos aperfeiçoarmos profissionalmente.

REFERÊNCIAS

[1] Byunghoon Kang, J. W. Shin and P. Park, "Piecewise linear motion blur identification using morphological filtering in frequency domain,"2009 ICCAS-SICE, Fukuoka, Japan, 2009, pp. 1928-1930.

[2] Wei-Guo He, Shao-Fa Li and Gui-Wu Hu, "Blur identification using an adaptive ADALINE network,"2005 International Conference on Machine Learning and Cybernetics, Guangzhou, China, 2005, pp. 5314-5317 Vol. 9, doi: 10.1109/ICMLC.2005.1527882.

[3] F. Qin et al., "Blind Image Restoration with Horizontal Motion Blur Based on Point Spread Function Estimation in Frequency Domain,"2022 14th International Conference on Computer Research and Development (ICCRD), Shenzhen, China, 2022, pp. 286-290, doi: 10.1109/ICCRD54409.2022.9730251

[4] M. A. Khan, S. A. Irtaza and A. Khan, "Detection of Blur and Non-Blur Regions using Frequency-based Multi-level Fusion Transformation and Classification via KNN Matting,"2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), Karachi, Pakistan, 2019, pp. 1-8, doi: 10.1109/MACS48846.2019.9024805.

[5] Sun, Wenfei Cao, Zongben Xu and J. Ponce, "Learning a convolutional neural network for non-uniform motion blur removal,"2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 769-777, doi: 10.1109/CVPR.2015.7298677.

[6] D. Gong et al., "From Motion Blur to Motion Flow: A Deep Learning Solution for Removing Heterogeneous Motion Blur,"2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 3806-3815, doi: 10.1109/CVPR.2017.405.

[7] K. -H. Liu, C. -H. Yeh, J. -W. Chung and C. -Y. Chang, "A Motion Deblur Method Based on Multi-Scale High Frequency Residual Image Learning,"in IEEE Access, vol. 8, pp. 66025-66036, 2020, doi: 10.1109/ACCESS.2020.2985220.

[8] C. Min, G. Wen, B. Li and F. Fan, "Blind Deblurring via a Novel Recursive Deep CNN Improved by Wavelet Transform,"in IEEE Access, vol. 6, pp. 69242-69252, 2018, doi: 10.1109/ACCESS.2018.2880279.

[9] O. Ronneberger, P. Fischer and T. Brox. "U-net: Convolutional networks for biomedical image segmentation,"in Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18 (pp. 234-241). Springer International Publishing.

[10] J. Redmon, S. Divvala, R. Girshick and A. Farhadi. "You only look once: Unified, real-time object detection,"in Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[11] "Flickr Image dataset,"in Kaggle. Available in `https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset?resource=download`