# Projeto

Jeffson Carneiro
Cin
UFPE
Recife, Brasil.

jcss3@cin.ufpe.br

*Abstract*— Esse documento tem como objetivo descrever o projeto que teve como base o pipeline do artigo[1].

*Keywords— deep learning, neural network, CNN (Convolutional neural networks), high dynamic range (HDR), autoencoder***.**

## I. Introdução

O ruído de imagem é uma questão comum nas fotografias digitais, e essa questão é ainda mais proeminente em sensores mais pequenos utilizados em dispositivos como smartphones ou câmeras de ação. Melhoria contínua no poder de processamento de pequenos dispositivos tem permitido muitos avanços na fotografia computacional, onde são utilizados algoritmos para compensar as propriedades físicas limitadas do sistema de imagem. Os algoritmos que visam a remoção do ruído digital são um tópico muito recorrente da literatura da fotografia computacional. A maioria deles utiliza apenas a informação armazenada na imagem de entrada (eles são geralmente referidos como algoritmos de "quadro único"), enquanto outros sugerem a utilização de informação armazenada em imagens adicionais. Esses "denoisers" multi-fotogramas fazem normalmente uso de medições de similaridade para combinar a informação de múltiplos pixels ou patches.[1]

Um algoritmo de “denoising” multi-quadro está no coração do sistema HDR+, sistema ao qual o artigo [1] foca e o qual este projeto se baseia.

## II. Projeto

Durante o tempo para ser desenvolvido foram pesquisadas diferentes abordagens sobre a arquitetura/design a ser usado em várias fontes[2][3] e ao fim foi escolhido a proposta de usar um autoencoder como a arquitetura principal.

### A. Objetivo

O sistema HDR+ tinha como objetivo produzir imagens individuais com bom contraste e alcance dinâmico, pouco ruído e borrão de movimento, e cores agradáveis na maioria dos cenários de filmagem, tudo isto com um aspecto natural e que requer pouca ou nenhuma contribuição do usuário.

Assim o objetivo desse projeto foi o de propor uma solução com Deep Learning que conseguisse construir imagens individuais com bom contraste e faixa dinâmica, pouco ruído e além de tudo isso que tivesse o aspecto natural com pouca interferencia do usuario.

Dito isto, foi decido usar um autoencoder para reconstruir as imagens de tal forma que a entrada seria a imagem do usuário e o resultado seria a reconstrução dessa imagem de forma que ela tivesse todas as qualidades citadas no paragrafo anterior.

## III. Dataset

Os dados usados para contruir a Rede Neural foram o Conjunto de dados HDR+ Burst Photography[4] o mesmo usado pelo artigo[1].

O conjunto de dados está alojado no Google Cloud e está disponível tanto para navegação (é necessária uma conta Google) como para download (anonimamente).

Por limitações da maquina na qual iria ser desenvolvido o projeto, decidimos usar o subconjunto de imagens (153 imagens). Todas estas fotografias têm uma qualidade técnica fotográfica razoavelmente boa, e cobrem uma gama diversificada de cenas, o que seria importante para a generalização do nosso modelo quando fosse necessario reconstruir imagens não vistas.

## IV. Arquitetura

### A. AutoEncoder

Os Autoencoders codificam os valores de entrada x usando uma função f. Em seguida, decodificam os valores codificados f (x) usando uma função g para criar valores de saída idênticos aos valores de entrada. O objetivo do Autoencoder é minimizar o erro de reconstrução entre a entrada e a saída. Isso ajuda os Autoencoders a aprender os recursos importantes presentes nos dados. [3]

Quando uma representação permite uma boa reconstrução de sua entrada, ela retém grande parte das informações presentes na entrada. Assim a ideia de usar os autoencoder foi baseada no seguinte pensamento: “se conseguirmos baixar a dimensionalidade do encoder até tal ponto onde o modelo aprenda as características mais importantes ignorando os ruídos teremos como reconstruir qualquer imagem sem tais ruídos e dependendo dos dados usados podemos até reconstruir as imagens com qualidades e aspectos diferentes.”

### B. Camadas Convulacionais

Uma Rede Neural Convolucional (ConvNet / Convolutional Neural Network / CNN) é um algoritmo de Aprendizado Profundo que pode captar uma imagem de entrada, atribuir importância (pesos e vieses que podem ser aprendidos) a vários aspectos / objetos da imagem e ser capaz de diferenciar um do

outro. O pré-processamento exigido em uma ConvNet é muito menor em comparação com outros algoritmos de classificação. Enquanto nos métodos primitivos os filtros são feitos à mão, com treinamento suficiente, as ConvNets têm a capacidade de aprender esses filtros / características.[5]

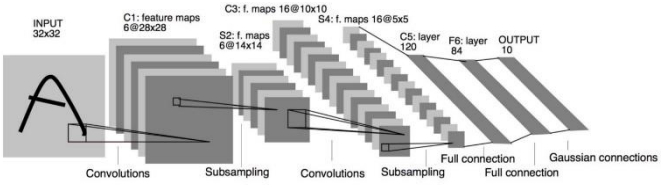


*Fig. 1. Exemplo de arquitetura convolucional.[6]*

As convoluções funcionam como filtros que enxergam pequenos quadrados e vão "escorregando" por toda a imagem captando os traços mais marcantes. Explicando melhor, com uma imagem 32x32x3 e um filtro que cobre uma área de 5x5 da imagem com movimento de 2 passos (chamado de stride), o filtro passará pela imagem inteira, por cada um dos canais, formando no final um feature map ou activation map de 28x28x1.[6]

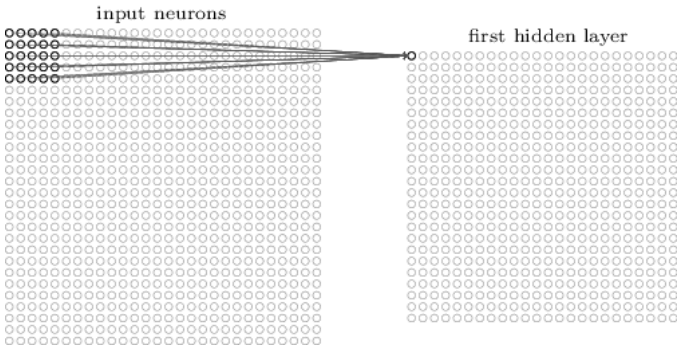


*Fig. 2. Exemplo de Convolução.[6]*

A profundidade da saída de uma convolução é igual a quantidade de filtros aplicados. Quanto mais profundas são as camadas das convoluções, mais detalhados são os traços identificados com o activation map. Além do tamanho do filtro e o stride da convolução como hiperparâmetro, quem está modelando uma CNN também tem que escolher como será o padding. O padding pode ser nenhum, no qual o output da convolução ficará no seu tamanho original, ou zero pad, onde uma borda é adicionada e preenchida com 0's. O padding serve para que as camadas não diminuam muito mais rápido do que é necessário para o aprendizado.

As camadas convolucionais foram usadas na parte do encoder e consitiam em camadas com filtros (2x2), funções de ativação relu e padding "same" onde o tamanho da imagem depois da convolução será igual ao do input assim completamos as bordas(Esquerda,direita,cima e baixo) com zeros.

### *C. Camadas Pooling*

As camadas pooling foram usadas para regularizar as imagens após as convoluções na parte do encoder, mais precisamente as camadas average pooling.
Uma camada de pooling recebe cada saída do mapa de características da camada convolucional e prepara um mapa de características condensadas.

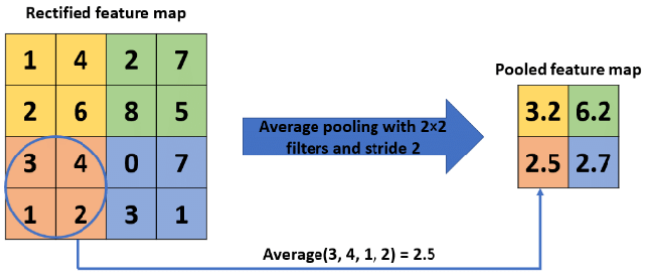


*Fig. 3. Exemplo Average Pooling 2x2.]*

### *D. Camadas Convulacionas Transpose*

Foram usadas na parte do decoder do autoencoder e são basicamente o inverso das camadas de convolução
A convolução transposta, também chamada de convolução fracionada ou deconvolução, serve para o proposito de aumentar a largura e altura de entrada.[7]
Em resumo:

- Em comparação com as convoluções que reduzem as entradas por meio de kernels, as convoluções transpostas transmitem as entradas.[7]
- Se uma camada de convolução reduz a largura e altura de entrada em "nw" e "hh" tempo, respectivamente. Então, uma camada de convolução transposta com os mesmos tamanhos de *kernel*, preenchimento e passos aumentará a largura e altura de entrada em "nw" e "hh", respectivamente.[7]
- Podemos implementar operações de convolução pela multiplicação da matriz, as convoluções transpostas correspondentes podem ser feitas pela multiplicação da matriz transposta.[7]

### *E. Camadas Upsampling*

A camada de Upsampling é uma camada simples sem pesos que duplicará as dimensões de entrada, basicamente o inverso das camadas pooling usadas no encoder a camada upsampling tem como objetivo aumentar as dimensões das imagens e foi usada no decoder.

### F. *Arquitetura Final*

O autoencoder possui 3 partes bastante importantes: o encoder , a representação latente e o decoder. No encoder foram usadas camadas convolucionais e pooling, a representação latente é o output do encoder, mais precisamente a ultima camada de pooling e o decoder foi formado com camadas convolucionais trabnspostas e upsampling tendo como camada final uma convolucional com filtro 3x3.
Durante o treinamento foram usados o optimizador Adam, função de perda "mse"(Mean Squared Error) e verificada a métrica de acurácia.

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img_input (InputLayer)      [(None, 320, 320, 3)]     0

 conv2d (Conv2D)             (None, 320, 320, 32)      416

 conv2d_1 (Conv2D)           (None, 320, 320, 32)      4128

 average_pooling2d (AverageP  (None, 160, 160, 32)     0
 ooling2D)

 conv2d_2 (Conv2D)           (None, 160, 160, 64)      8256

 conv2d_3 (Conv2D)           (None, 160, 160, 64)      16448

 average_pooling2d_1 (Averag  (None, 80, 80, 64)       0
 ePooling2D)

 conv2d_4 (Conv2D)           (None, 80, 80, 128)       32896

 conv2d_transpose_1 (Conv2DT  (None, 80, 80, 64)       32832
 ranspose)

 up_sampling2d (UpSampling2D  (None, 160, 160, 64)     0
 )

 conv2d_transpose_2 (Conv2DT  (None, 160, 160, 32)     8224
 ranspose)

 conv2d_transpose_3 (Conv2DT  (None, 160, 160, 32)     4128
 ranspose)
```

*Fig. 4. Parte 1 do Sumario da Rede*

```
 up_sampling2d_1 (UpSampling  (None, 320, 320, 32)     0
 2D)

 Decoder_Output (Conv2D)     (None, 320, 320, 3)       867

=================================================================
Total params: 108,195
Trainable params: 108,195
Non-trainable params: 0
_________________________________________________________________
```

*Fig. 5. Parte 2 do Sumario da Rede*

## V. Ferramentas

O ambiente utilizada para o Desenvolvimento do trabalho foram:

- Jupyter Notebook (Maquina Pessoal)
- Colab (Usar GPU)

As tecnologias usadas para o Desenvolvimento em si foram:

- Linguagem Python
- Tensorflow Criação da rede neural
- OpenCV para vizualização

## VI. Avaliação

Usamos algumas métricas (MSE e Acurácia) para acompanharmos a evolução da rede durante o treinamento além de fazer comparações das imagens reconstruídas com as originais e algumas feitas pelo algoritmo proposto do artigo[1].

### A. *Metricas*

- Loss

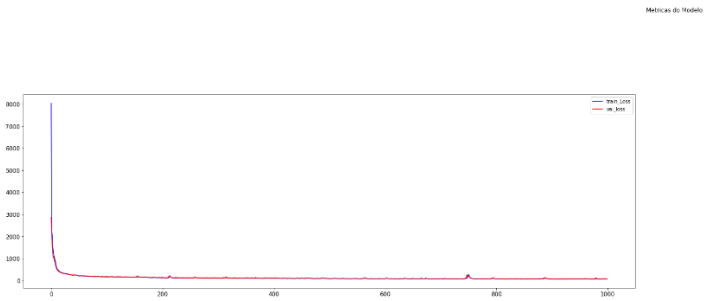

*Fig. 6. Gráfico que representa a perda (MSE) durante treinamento(azul) e validação(vermelho).*

- Acurácia

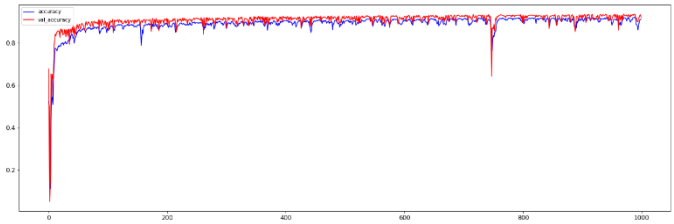

*Fig. 6. Gráfico que representa acurácia durante treinamento(azul) e validação(vermelho).*

### B. *Comparações de Imagens*

*Fig. 7. Comparação entre a imagem construída pelo autoencoder(imagem da esquerda) e a imagem usada durante o treinamento(imagem mais a direita).*

*Fig. 8. Verificando como o modelo se comporta com ruídos. Comparação entre a imagem construída pelo autoencoder(imagem da esquerda) e a imagem de entrada (imagem mais a direita).*

*Fig. 9. Verificando como o modelo se compara com os resultados obtidos pelo algoritmo do artigo[1]. Comparação entre a imagem construída pelo autoencoder(imagem da direita) e a imagem do artigo(imagem mais a esquerda). Ambos são os resultados a partir de uma imagem normal.*

*Fig. 10. Verificando como o modelo se compara com os resultados obtidos pelo algoritmo do artigo[1] em uma imagem com movimento. Comparação entre a imagem construída pelo autoencoder(imagem da direita) e a imagem do artigo(imagem mais a esquerda). Ambos são os resultados a partir de uma imagem normal.*

## VII. Conclusão

Apesar do autoencoder conseguir uma boa contrução da imagem passada, ainda assim não conseguiu entregar uma qualidade melhor do que no artigo, ou seja, ele não conseguiu melhorar, consideravelmente, o nivel de qualidade, detalhes e aspecto natural da imagem passada.

## References


[1] https://arxiv.org/pdf/2110.09354.pdf
[2] https://arxiv.org/pdf/1710.07480.pdf
[3] https://www.deeplearningbook.com.br/principais-tipos-de-redes-neurais-artificiais-autoencoders/
[4] https://hdrplusdata.org/
[5] https://www.deeplearningbook.com.br/introducao-as-redes-neurais-convolucionais/
[6] https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184
[7] https://pt.d2l.ai/chapter_computer-vision/transposed-conv.html