UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GABRIEL VIEIRA FERREIRA

DETECÇÃO *SEAM CARVING* UTILIZANDO *DEEP LEARNING*

BAURU
Dezembro/2020

GABRIEL VIEIRA FERREIRA

DETECÇÃO *SEAM CARVING* UTILIZANDO *DEEP LEARNING*

Trabalho de Conclusão de Curso de Bacharel em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Kelton Augusto Pontara da Costa

BAURU
Dezembro/2020

Gabriel Vieira Ferreira Detecção *Seam Carving* Utilizando *Deep Learning* / Gabriel Vieira Ferreira. – Bauru, Dezembro/2020- 40 p. : il. (algumas color.) ; 30 cm.
Orientador: Prof. Dr. Kelton Augusto Pontara da Costa
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Ciência da Computação, Dezembro/2020.

Gabriel Vieira Ferreira

Detecção *Seam Carving* Utilizando *Deep Learning*

Trabalho de Conclusão de Curso de Bacharel em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. Kelton Augusto Pontara da Costa
Orientador
Departamento de Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho

Profª. Dra. Simone das Graças Domingues Prado
Departamento de Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Prof. Dr. Aparecido Nilceu Marana
Departamento de Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Bauru, 14 de Dezembro de 2020.

Dedico este trabalho a Deus, sem Ele nada seria possível. A minha família que sempre me apoiou. Ao Departamento de Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho" campus Bauru, pois me mostrou como se dedicar a uma causa maior com o coração e a todos amigos e pessoas que passam pela minha vida para engrandecer e auxiliar de diversas formas.

Agradecimentos

Agradeço a Deus, pois sem Ele não teria nem o ar que respiro, quem dirá forças diariamente para enfrentar esses quatro anos. Acredito que Ele coloca muitas pessoas especiais em nossas vidas, das quais também agradeço por toda ajuda, a toda minha família, professores e amigos.

"Privacidade não é sobre ter algo a esconder. É sobre ter algo para proteger. E esse algo é quem você é. É algo em que você acredita. É quem você quer se tornar. Privacidade é o direito de si mesmo. É o que lhe permite compartilhar com o mundo quem você é nos seus próprios termos."

(Edward Snowden)

Resumo

A fraude em imagens se apresenta com uma questão de suma importância a ser combatida, pois ela possibilita a existência de um alto risco de violação de direitos autorais e aquisição ilegal. *Seam Carving* é um método de redimensionamento capaz de modificar a largura ou altura de imagens sendo sensível ao seu conteúdo. O objetivo central do presente trabalho é encontrar um modelo capaz de detectar com precisão fraudes em imagens causada pelo *Seam Carving* utilizando um método *Deep Learning* específico para trabalhar com imagens denominado *Convolutional Neural Network (CNN)*. Os resultados foram ótimos, pois o método obteve 99% de acurácia, ou seja, acerto sobre a classificação de adulteração ou não na imagem, comparando-se a pesquisas relacionadas. A técnica *Deep Learning* em questão mostrou-se eficaz para prevenção e detecção de fraudes em imagens.
Palavras-chave: *Seam Carving*, *CNN*, detecção de fraudes.

Abstract

Image fraud presents itself as a matter of paramount importance to be tackled, as it allows for the existence of a high risk of copyright infringement and illegal acquisition. Seam Carving is a resizing method capable of modifying the width or height of images while being sensitive to their content. The main objective of this work is to find a model capable of accurately detecting image fraud caused by Seam Carving using a specific Deep Learning method to work with images called Convolutional Neural Network (CNN). The results were excellent, as the method obtained 99 % accuracy, that is, correct on the classification of adulteration or not in the image, compared to related research. The Deep Learning technique in question proved to be effective for preventing and detecting image fraud.
Keywords: Seam Carving, CNN, fraud detection.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

CNN	*Convolutional Neural Network*
BMP	*Windows Bitmap*
ELM	*Extreme Learning Machine*
GAN	*Generative Adversarial Network*
JPEG	*Joint Photographics Experts Group*
LBP	*Local Binary Pattern*
PSO	*Patch-based Sobel Operator*

Sumário

1 Introdução

O fácil acesso a dispositivos de imagem digital (câmeras digitais, filmadoras digitais ou *smartphones*) e o desempenho dos mesmos melhorou de maneira significativa. Consequentemente, as imagens digitais podem ser facilmente manipuladas por usuários não profissionais (CHOI; LEE; LEE, 2013). Portanto, existe um alto risco de violação de direitos autorais e manipulação ou aquisição ilegal de imagens. Este trabalho aborda esses problemas, utilizando de técnicas *Deep Learning* com objetivo de detectar fraudes em imagens, mais especificamente, utilizando uma *Convolutional Neural Networks (CNN)* para detectar adulterações em imagens causadas pelo método *Seam Carving*.

Três métodos principais compõem a base deste projeto. O método denominado *Seam Carving* proposto por Avidan e Shamir (2007) baseia-se no acúmulo de energia para redimensionamento de imagens, ou seja, cria uma descontinuidade do conteúdo da imagem e uma remoção ou adição de costuras (*seams*) através de funções de energia que contêm apenas as informações de gradiente (LIN et al., 2016). Ao remover uma costura específica, é distribuída a energia de cada *pixel* na costura através dos *pixels* conectados em oito, ou seja, *pixels* vizinhos conetados na horizontal, vertical e diagonal, a fim de evitar a extrema concentração de costuras. *Local Binary Pattern (LBP)* proposto por He e Wang (1990) é um operador de textura simples e muito eficiente, que rotula os *pixels* de uma imagem limitando a vizinhança de cada *pixel* e considerando o resultado como um número binário (PIETIKINEN et al., 2011). A propriedade mais importante do operador *LBP* em aplicações do mundo real é sua invariância contra alterações monotônicas do nível de cinza causadas, por exemplo, pelas variações de iluminação, além da simplicidade computacional, que possibilita analisar imagens em ambientes desafiadores em tempo real. O objetivo de uma *CNN* é aprender características de ordem superior nos dados por meio de convoluções. Essa rede é adequada para reconhecimento de objetos e classificação de imagens, também é adequada à análise de texto por meio do reconhecimento óptico de caracteres, especialmente sendo úteis ao analisar palavras como unidades textuais distintas. As *CNNs* também são eficazes em analisar som (PATTERSON; GIBSON, 2017). Patterson e Gibson (2017) descrevem todo o funcionamento de uma *CNN*. Quando a imagem é alimentada em um computador, ele basicamente a converte em uma matriz de valores de *pixel*. Os valores de *pixel* variam de 0 a 255, e as dimensões desta matriz serão de [largura da imagem x altura da imagem x número de canais], essa imagem é então denominada de imagem digital. Uma imagem em tons de cinza tem um canal e as imagens coloridas têm três canais: vermelho, verde e azul (*red, green and blue - RGB*), por exemplo, uma imagem de entrada colorida com uma largura de 11 e uma altura de 11, ou seja, 11 x 11, possui dimensão de matriz de [11 x 11 x 3]. Portanto, tem-se uma matriz 3D.

Existem vários métodos propostos para detectar falsificações baseadas em *Seam Carving*,

no entanto, tal detecção sob ataques de recompressão em imagens *Joint Photographics Experts Group (JPEG)* não foi explorada (LIU, 2016). O mesmo autor propôs um método híbrido de detecção baseado em mineração de recursos em larga escala para distinguir as imagens *JPEG* adulteradas das imagens *JPEG* intocadas, sob ataques de recompressão. O aprendizado de conjunto (*Ensemble Learning*) foi adotado para lidar com a alta dimensionalidade e reconhecer os padrões de imagens intocadas e extraídas de imagens recomprimidas com a mesma ou menor qualidade.

Recentemente, foram publicados trabalhos com foco em métodos de detecção *Seam Carving* aplicando técnicas inteligentes, a fim de obter melhores taxas para tais detecções. O primeiro executou quatro tarefas para detecção *Seam Carving*: um novo *Patch-based Sobel Operator (PSO)*, implementando o método *PSO* na *Extreme Learning Machine (ELM)*, o *Jury Voting Scheme* e, finalmente, a utilização da *Combinatorial Fusion Technique* (CHENG et al., 2018). Um segundo trabalho realizado propôs detectar adulterações de imagem usando *Convolutional Neural Networks (CNN)* e *Local Binary Patterns (LBP)*, com foco em *Seam Carving* e *Seam Insertion* (CIESLAK; COSTA; PAPA, 2018). Vale ressaltar que ambos os trabalhos obtiveram resultados promissores ao utilizar técnicas inteligentes para detecção *Seam Carving* com aproximadamente 99% e 98% de acurácia em casos específicos, respectivamente. Outro trabalho propôs um classificador multiclasse que possui quatro classes de saídas: 0% ou sem entalhe na costura, 10% entalhado na costura, 20% entalhado na costura e 40% entalhado na costura, o modelo de rede utilizada pelos autores obteve cerca de 84% de precisão (NAZARİ; AKGÜN, 2020). Por fim, outro trabalho consistiu em utilizar características mais eficazes por meio da otimização conjunta de extração de características e classificação de padrões (YE et al., 2019). Os dois últimos trabalhos citados também utilizaram *LBP* e técnicas inteligentes para detecção de adulteração em imagens causada pelo *Seam Carving*.

A detecção *Seam Carving* é útil para evitar fraudes e imagens geradas artificialmente, frequentemente usadas para falsificação e outros tipos de crimes digitais. Portanto, desenvolver métodos eficientes para sua detecção é uma grande contribuição para a comunidade científica. Como visto no parágrafo anterior, muitos trabalhos estão sendo desenvolvidos utilizando técnicas inteligentes como auxiliares para resolução de fraudes em imagens. Portanto, o objetivo central e a metodologia do presente trabalho é utilizar o método *Seam Carving* para fraudar imagens e posteriormente utilizar uma técnica inteligente denominada *CNN* para detectar se a imagem foi fraudada ou não, e qual o tipo de fraude ocorreu, *Seam Carving* (remoção de *seams*) ou *Seam Insertion* (inserção de *seams*). O objetivo secundário é contribuir para comunidade científica através de um modelo de rede para detecção de fraudes em imagens. Os resultados obtidos foram ótimos se comparado a trabalhos relacionados, atingindo 99% de acurácia.

O restante deste trabalho está organizada da seguinte forma. As Seções 2.1, 2.2 e 2.3 apresentam os métodos e respectivas operações envolvidas. A Seção 3 apresenta a metodologia

utilizada nesta pesquisa. A Seção 4 apresenta os resultados obtidos, e por fim a Seção 5 apresenta as conclusões finais obtidas pelo autor.

2 Fundamentação Teórica

Está Seção tem por objetivo apresentar a fundamentação teórica utilizada neste trabalho, na Seção 2.1 é apresentado o método *Seam Carving*, na Seção 2.2 o método *LBP*, por fim na Seção 2.3 o método *CNN*.

2.1 *Seam Carving*

Seam Carving é um operador de imagem simples que oferece suporte ao redimensionamento de imagem com reconhecimento de conteúdo para redução e expansão. Uma costura é um caminho ideal de *pixels* conectado em oito (com os *pixels* vizinhos nas horizontais, verticais e diagonais) em uma única imagem de cima para baixo, ou da esquerda para a direita, ambos mostrados na Figura 1, onde a otimização é definida por uma função de energia da imagem. Ao retirar ou inserir costuras repetidamente em uma direção, pode-se alterar a proporção de uma imagem. Aplicando esses operadores em ambas as direções, pode-se redimensionar a imagem para um novo tamanho. A seleção e a ordem das costuras protegem o conteúdo da imagem, conforme definido pela função de energia. O método *Seam Carving* também pode ser usado para aprimorar o conteúdo da imagem e remover objetos (AVIDAN; SHAMIR, 2007).

Figura 1 – Exemplo de costura (*seam*) ocasionada pelo *Seam Carving*.

Fonte: Adaptado de Avidan e Shamir (2007).

A questão em que o método se envolve é em como escolher os *pixels* a serem removidos. O objetivo central é remover *pixels* imperceptíveis que se misturam com seus vizinhos, para realização de tal objetivo é necessário conhecer o conceito de função de energia, representado por e descrito através da Equação 2.1, onde I é a imagem (AVIDAN; SHAMIR, 2007).

$$e(I) = \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right| \tag{2.1}$$

Dada uma função de energia, Avidan e Shamir (2007) demonstraram uma séries de outras técnicas que não se apresentaram tão eficientes, por exemplo, remover os *pixels* com a menor energia em ordem crescente, isso destrói a forma retangular da imagem, porque pode-se remover um número diferente de *pixels* de cada linha; Ou remover *pixels* e recortar colunas inteiras com a energia mais baixa, porém mesmo assim podem aparecer resultantes negativos na imagem. Portanto, foi necessário desenvolver um operador de redimensionamento menos restritivo do que o corte ou a remoção da coluna, mas que preserve o conteúdo da imagem. Isso originou o conceito de costuras internas ou *seams*, definida pela letra s na Equação 2.2, onde i corresponde à coordenada da linha e $x(i)$ corresponde à coordenada da coluna em i.

$$s^x = \sum_{i=1}^{n} s_i^x = \{x(i), i\}_{i=1}^{n}, \quad \forall i, |x(i) - x(i-1)| \leq 1 \tag{2.2}$$

Formalmente, imaginando a imagem I apresentada na Equação 2.1 como uma matriz $n \times m$, onde n é o número de linhas e m o número de colunas. A variável x descrita na Equação 2.2 representa um mapeamento $x : [1, ..., n] \rightarrow [1, ..., m]$. Ou seja, uma costura vertical é um caminho de *pixels* conectado em oito na imagem de cima para baixo, contendo um, e apenas um, *pixel* em cada linha da imagem. O processo de remoção de linhas é muito semelhante, basta trocar os eixos linha-coluna que são usados como parâmetros nas equações apresentadas.

O impacto visual é perceptível apenas ao longo do caminho da costura, deixando o resto da imagem intacta e a restrição $|x(i) - x(i-1)| \leq 1$ da Equação 2.2 pode ser substituída por $|x(i) - x(i-1)| \leq k$, e obter uma coluna simples (ou linha) para $k = 0$, um conjunto de *pixels* conectado por partes ou até mesmo completamente desconectado para qualquer valor pertencente ao intervalo $1 \leq k \leq m$.

Dada uma função de energia e, pode-se definir o custo das costuras como $E(s) = E(I_s) = \sum_{i=1}^{n} e(I(s_i))$, o algoritmo procura a costura ideal s^* que minimiza esse custo das costuras.

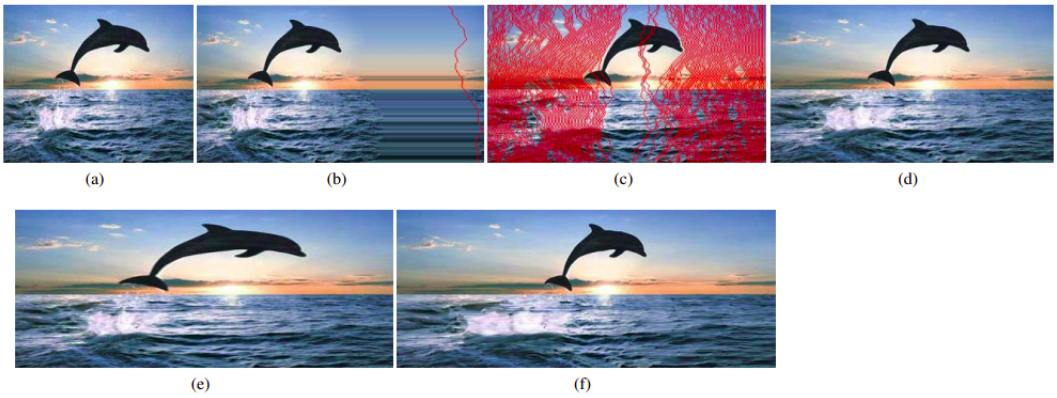
$$s^* = \min_s E(s) = \min_s \sum_{i=1}^{n} e(I(s_i)) \tag{2.3}$$

A costura ideal pode ser encontrada usando a programação dinâmica. A primeira etapa é atravessar a imagem da segunda linha para a última linha e calcular a energia mínima cumulativa M para todas as costuras conectadas possíveis para cada entrada (i, j), isto é descrito na Equação 2.4. No final deste processo, o valor mínimo da última linha em M indicará o final da costura vertical mínima conectada.

$$M(i, j) = e(i, j) + min(M(i-1, j-1), M(i-1, j), M(i-1, j+1)) \tag{2.4}$$

Na Figura 2 é possível analisar a eficácia do *Seam Carving* ao redimensionar uma imagem. Em (a) tem-se a imagem original, em (b) mostra a situação onde o algoritmo restringiu-se em inserir somente a costura ideal (a mesma costura), por isso o método *Seam Carving* não faz essa operação. As inserções das costuras na ordem de remoção são mostradas em (c), as mesmas inserções atingiram o aumento de 50% desejado em (d). Além disso, os autores utilizaram mais uma etapa de inserções de costura de 50% em (e) e (f), os resultados em (f) foram visualmente melhores.

Figura 2 – Inserção da costura.



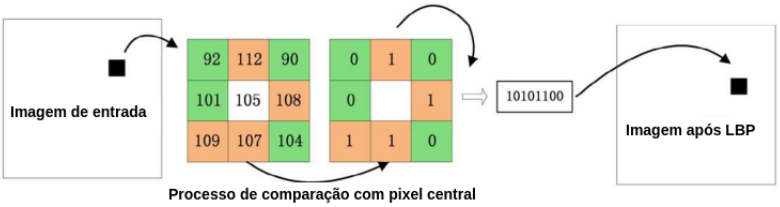
Fonte: Avidan e Shamir (2007).

2.2 *Local Binary Patterns*

O operador *Local Binary Patterns (LBP)* original/básico rotula os *pixels* de uma imagem com números decimais, chamados de padrões binários locais ou códigos *LBP*, que codificam a estrutura local em torno de cada *pixel* (HUANG et al., 2011), o procedimento do *LBP* está ilustrado na Figura 3. Seu funcionamento se inicia quando cada *pixel* é comparado com seus oito vizinhos e subtraindo o valor do *pixel* central, no caso de uma vizinhança 3×3, pois esse valor pode ser alterado. Os valores estritamente negativos resultantes são codificados com 0 e os outros com 1. Um número binário é obtido pela concatenação de todos esses códigos binários no sentido horário, começando no canto superior esquerdo, e seu valor decimal correspondente é usado para rotulagem.

O processo *LBP* também pode ser descrito através da Equação 2.5, onde g_c é o valor do *pixel* central, g_p é o valor de seus vizinhos, P é o número total de vizinhos envolvidos e R é o raio da vizinhança. No exemplo da Figura 3, R é igual a 1 e P igual 8, o que significa que os *pixels* vizinhos conectados em oito estão envolvidos no cálculo do *LBP*. Em seguida, o resultado desse cálculo é codificado em um valor inteiro de 8 bits. Assim, a imagem de entrada

Figura 3 – Funcionamento básico do *LBP*.



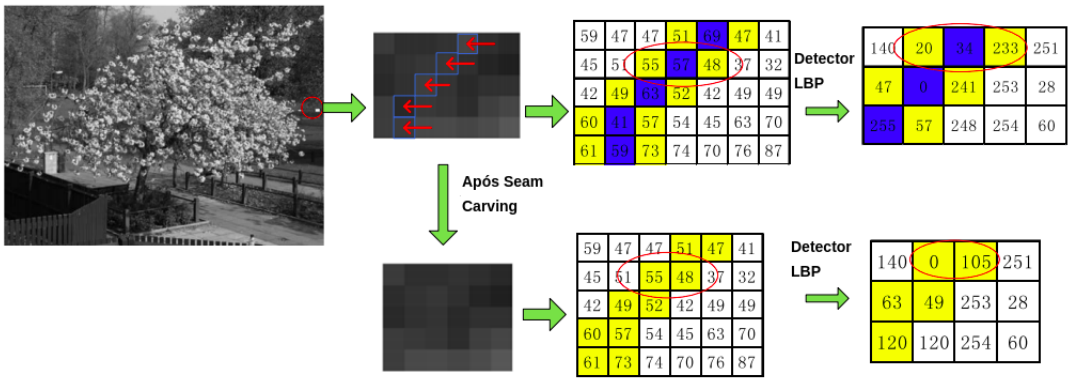
Fonte: Adaptado de Yin et al. (2015).

é transformada em outra imagem por cálculo do valor *LBP* de cada *pixel*.

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^P, \quad s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{2.5}$$

Quando uma costura vertical é removida, todos os *pixels* em seu lado direito são deslocados para a esquerda para preencher a lacuna do caminho removido. Para os *pixels* exatamente adjacentes a essa junção, seus *pixels* vizinhos com oito conexões serão significativamente alterados. Portanto, seus códigos *LBP* serão bastante diferentes dos originais (YIN et al., 2015). A Figura 4 mostra esse exemplo das alterações de *LBP* quando uma costura é removida.

Figura 4 – Exemplos de influência *LBP* para remover um *pixel*.



Fonte: Adaptado de Yin et al. (2015).

Na Figura 4 um bloco 5×7 na imagem original é ampliado para ilustrar a influência da remoção de uma emenda em direção aos *LBPs* desses *pixels* adjacentes a esta emenda. A linha superior mostra os valores de *pixel* do bloco original e seus *LBPs*. Os *pixels* que formam a costura a ser removida são marcados em azul. Os valores *LBP* dos *pixels* no bloco interno 3×5 também são mostrados. Após a remoção da emenda, o bloco 5×7 se transforma em um bloco 5×6 porque um *pixel* é removido em cada linha. Os valores *LBP* de seu bloco interno

3×4 também são mostrados. Por observação é concluído que quando uma emenda é removida de uma imagem, há mudanças significativas para os valores de *LBP* dos *pixels* vizinhos ao longo da emenda.

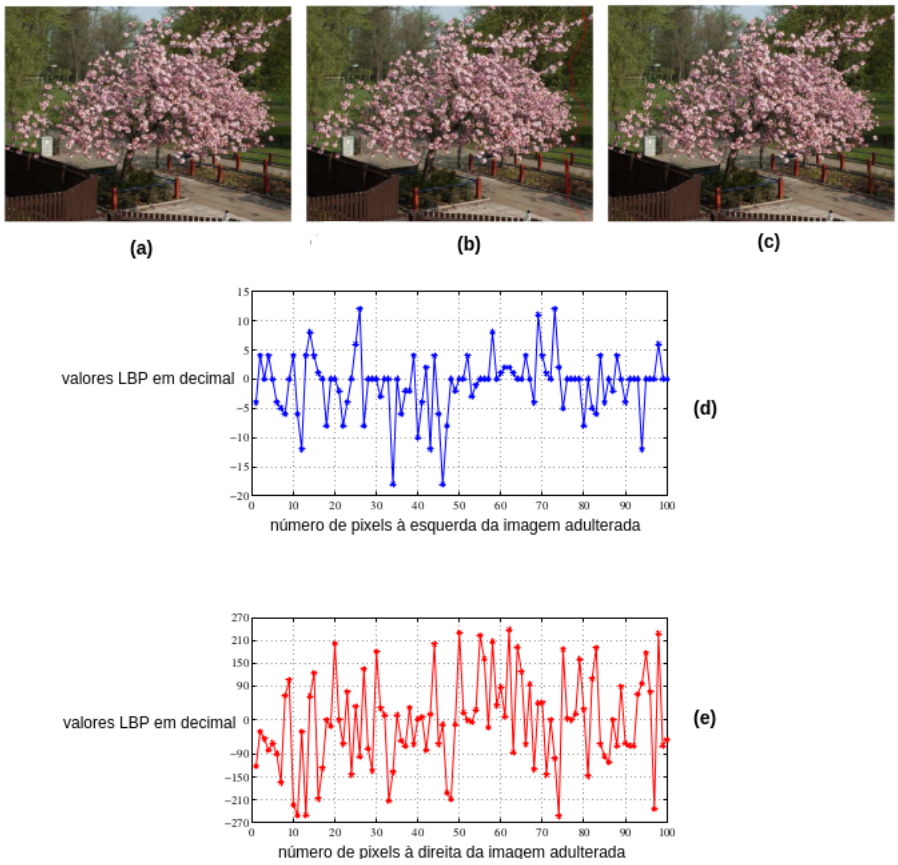
A influência da formação de costuras em relação aos valores *LBP* dos *pixels* vizinhos ao longo da costura é mostrado na Figura 5. Em (a) tem-se a imagem original, em (b) uma costura vertical mais precisamente no lado direito da imagem e em (c) a imagem após a remoção de uma costura. Para cada *pixel* na costura vertical existem duas colunas de *pixels* que são adjacentes à costura, nos lados esquerdo e direito, como apresentado no segmento superior da Figura 4 através das duas colunas de *pixels* em amarelo ao redor da coluna de *pixels* em azul. Em (d) é apresentado a diferença dos valores *LBP* para os *pixels* no lado esquerdo da costura removida entre a imagem original e a imagem adulterada. Em (e) a diferença dos valores *LBP* para os *pixels* no lado direito da costura removida entre a imagem original e a imagem adulterada. É aparente que há mudanças significativas para os valores *LBP* dessas duas colunas depois que uma emenda é removida de uma imagem (YIN et al., 2015). Assim, estes autores concluíram que quando uma emenda é removida de uma imagem, há mudanças significativas nos valores de *LBP* dos *pixels* vizinhos ao longo dessa emenda. Portanto, o *LBP* é sensível à alteração da textura das imagens causadas pelo método *Seam Carving*, especialmente aqueles *pixels* ao longo da costura removida.

2.3 *Convolutional Neural Network*

Convolutional Neural Network (CNN) é um método de aprendizado profundo mais comumente usado para tarefas relacionadas a imagens, como reconhecimento de imagens, detecção de objetos, segmentação de imagens e muito mais. As aplicações das *CNNs* são infinitas, abrangendo desde a visão em carros autônomos até a marcação automática de amigos em fotos no Facebook. Embora as *CNNs* sejam amplamente usadas para conjuntos de dados de imagens, também podem ser aplicadas a conjuntos de dados textuais (RAVICHANDIRAN, 2019). As *CNNs* podem identificar rostos, indivíduos, placas de rua e muitos outros aspectos dos dados visuais. As *CNNs* também são eficazes à análise de texto por meio do reconhecimento óptico de caracteres, mas são úteis ao analisar palavras como unidades textuais distintas e à análise de som. A eficácia dessas redes no reconhecimento de imagem é um dos principais motivos pelos quais o mundo reconhece o poder do aprendizado profundo (PATTERSON; GIBSON, 2017). A Figura 6 ilustra o funcionamento geral de uma *CNN*. Apenas como curiosidade, a inspiração biológica para *CNNs* é o córtex visual dos animais. As células do córtex visual são sensíveis a pequenas sub-regiões de entrada, denominadas de campo visual (ou campo receptivo).

Para melhor representação e fins de compreensão, neste trabalho, considera-se uma imagem em tons de cinza como matriz de entrada (RAVICHANDIRAN, 2019). Como a imagem em tons de cinza possui apenas um canal, é obtido uma matriz 2D, como ilustra a Figura 7.

Figura 5 – Mudança dos valores *LBP* após remoção de uma costura.



Fonte: Adaptado de Yin et al. (2015).

O objetivo da *CNN* nesta imagem é reconhecer que há um cavalo na fotografia, para tal as *CNNs* consistem minimamente em três camadas fundamentais: camada convolucional, camada de *pooling*, camada totalmente conectada (camada de classificação).

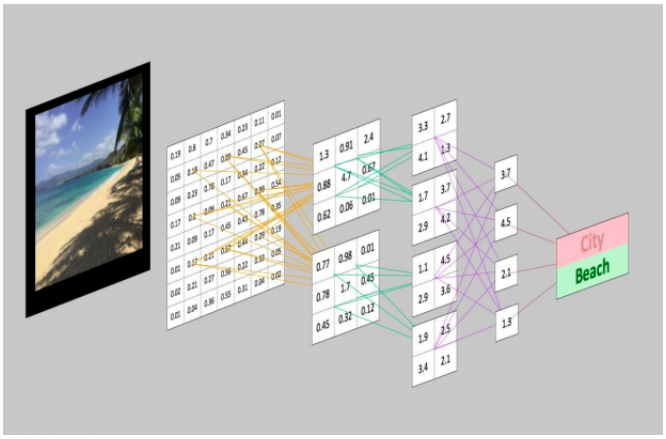
Além disso, as *CNNs* transformam os dados de entrada da camada de entrada por meio de todas as camadas conectadas em um conjunto de pontuações de classe fornecidas pela camada de classificação (PATTERSON; GIBSON, 2017). Existem muitas variações da arquitetura *CNN*, mas elas são baseadas no padrão de camadas, conforme mostra a Figura 8.

2.3.1 Camada convolucional

Possivelmente as características que ajudam a entender que esta é a imagem de um cavalo serão a estrutura corporal, rosto, pernas, cauda e assim por diante e para encontra-las a *CNN* usa o conceito de convolução que ocorre na camada convolucional, uma *CNN* pode ter várias Camada Convolucionais.

Cada imagem de entrada é representada por uma matriz de valores de *pixel*. Além da matriz de entrada, tem-se outra matriz chamada matriz de filtro ou *kernel*, mostrada na Figura

Figura 6 – *CNNs* e visão computacional.



Fonte: Patterson e Gibson (2017).

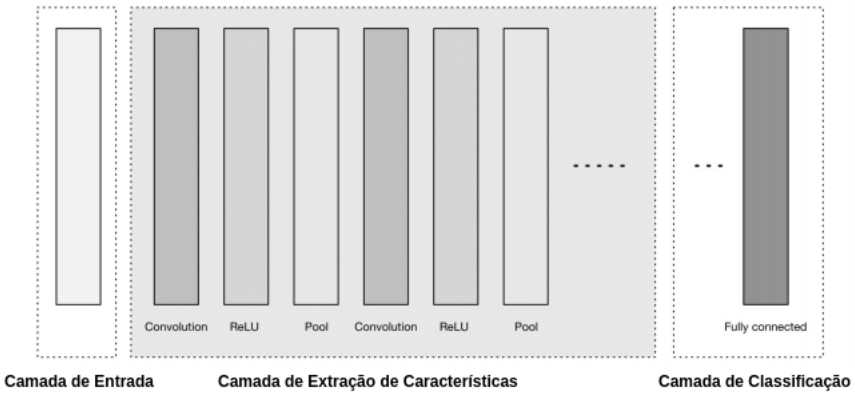
Figura 7 – Dados de entrada para uma *CNNs*.

Fonte: Adaptado de Ravichandiran (2019).

9, onde x_i, refere-se ao *pixel* da posição i na matriz de entrada, e w_i o equivalente na matriz de filtro.

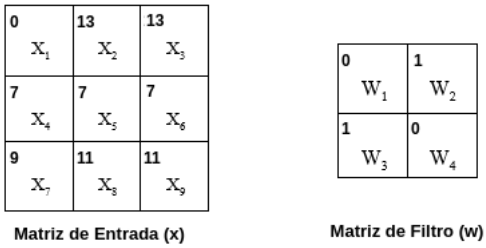
A matriz de filtro é deslizada sobre a matriz de entrada por um *pixel*, executa-se a multiplicação por elemento, soma-se os resultados obtendo-se um único número. Tal operação em toda a matriz de entrada resulta em uma nova matriz denominada mapa de características (*feature map*) ou mapa de ativação (*activation map*) ilustrado na Figura 10. Isso é chamado de operação de convolução, derivada da Equação 2.6 onde o_{ij} representa a saída que formará a matriz convolvida na linha i e coluna j, em I temos uma imagem de entrada, com largura W, e o filtro de tamanho $p \times q$. As operações realizadas foram $(0 \cdot 0 + 13 \cdot 1 + 7 \cdot 1 + 7 \cdot 0) = 20; (13 \cdot 0 + 13 \cdot 1 + 7 \cdot 1 + 7 \cdot 0) = 20; (7 \cdot 0 + 7 \cdot 1 + 9 \cdot 1 + 11 \cdot 0) = 16; (7 \cdot 0 + 7 \cdot 1 + 11 \cdot 1 + 11 \cdot 0) = 18$, formando a matriz convolvida a direta na Figura 10. Assim que a operação de convolução é realizada, alimentamos o resultado o_{ij}, para uma rede sem realimentação (*feedforward*) f para

Figura 8 – Arquitetura geral de *CNN* de alto nível.



Fonte: Adaptado de Patterson e Gibson (2017).

Figura 9 – Matriz de entrada e matriz de filtro.



Fonte: Adaptado de Ravichandiran (2019).

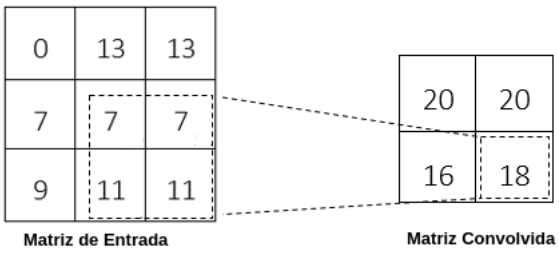
prever a saída $\hat{y}_i$, como mostra a Equação 2.7.

$$o_{ij} = \sum_{m=0}^{p-1} \sum_{n=0}^{q-1} W_{m,n} \cdot I_{i+m,j+n} \tag{2.6}$$

$$\hat{y}_i = f(o_{ij}) \tag{2.7}$$

A Figura 11 mostra a imagem real (a imagem de entrada) e a imagem convolvida (o mapa de características). É possível ver que o filtro detectou as bordas da imagem real como uma característica. Vários filtros são usados para extrair diferentes características da imagem. Portanto, em vez de usar um filtro, pode-se usar vários filtros para produzir vários mapa de características. Assim, a profundidade do mapa de características será o número de filtros. Ao usar sete filtros para extrair diferentes características da imagem, a profundidade do mapa de características será sete. Os valores ótimos da matriz de filtro, com os quais pode-se extrair características importantes das imagens, serão aprendidos por retropropagação, melhor

Figura 10 – Operação de convolução.



Fonte: Adaptado de Ravichandiran (2019).

detalhado na Seção 2.3.4. No entanto, precisa-se especificar o tamanho do filtro e o número de filtros que será usado.

Figura 11 – Resultado prático na imagem após processo de convolução.

Fonte: Adaptado de Ravichandiran (2019).

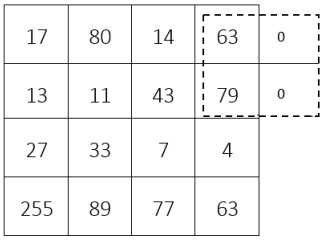
É válido reforçar dois conceitos importante dentro da camada convolucional, ou seja, os conceitos de *stride* e *padding*. É possível deslizar sobre a matriz de entrada qualquer número de *pixels*. O número de *pixels* que é deslizado sobre a matriz de entrada pela matriz de filtro é denominado de passo (*stride*). Se o *stride* é definido em dois, será deslizado dois *pixels* sobre a matriz de entrada com a matriz de filtro. Quando o *stride* é definido como um número pequeno, pode-se codificar uma representação mais detalhada da imagem do que quando o *stride* é definido como um número grande. No entanto, uma passada com um valor alto leva menos tempo para ser calculada do que uma com um valor baixo.

Na operação de convolução, em alguns casos, o filtro não se ajusta perfeitamente à matriz de entrada. Por exemplo, é realizado uma operação de convolução com *stride* igual a 2. Existe uma situação em que, quando move-se a matriz de filtro em dois *pixels*, ela atinge a borda e a matriz de filtro não se ajusta à matriz de entrada. Ou seja, alguma parte da matriz de filtro está fora da matriz de entrada.

Nesse caso é realizada a ação de preenchimento (*padding*), pode-se preencher a matriz de entrada com zeros para que o filtro possa se ajustar à matriz de entrada, essa ação é denominada de preenchimento de zero ou *zero padding*, conforme mostrado na Figura 12. Em

vez de preenchê-los com zeros, também pode-se descartar a região da matriz de entrada onde o filtro não se encaixa, ou seja, preenchimento válido ou *valid padding*.

Figura 12 – *Zero padding* na matriz de entrada.



Fonte: Ravichandiran (2019).

2.3.2 Camada de *pooling*

A operação de convolução resulta em mapas de características, porém estes são muito grandes em dimensão. Para reduzir as dimensões desses mapas, realiza-se uma operação de *pooling* ou agrupamento. Isso reduz as dimensões dos mapas de características e mantém apenas os detalhes necessários para que a quantidade de processamento seja reduzida. Por exemplo, para reconhecer um cavalo da Figura 7, precisa-se extrair e manter apenas as características do cavalo, descartando características indesejados como o plano de fundo da imagem e outros. Uma operação de *pooling* também é chamada de operação de *downsampling* ou *subampling*.

A operação de *pooling* não muda a profundidade dos mapas de características e afeta apenas a altura e a largura. Existem diferentes tipos de operações de agrupamento, incluindo agrupamento máximo (*max pooling*), agrupamento Médio (*average pooling*) e agrupamento de soma (*sum pooling*).

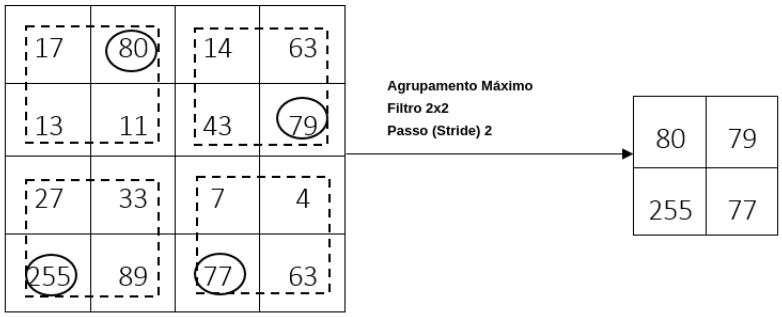
No agrupamento máximo, desliza-se sobre o filtro na matriz de entrada e simplesmente pega-se o valor máximo da janela de filtro, conforme mostrado na Figura 13.

No agrupamento médio, pega-se o valor médio da matriz de entrada dentro da janela de filtro. No agrupamento de soma, soma-se todos os valores da matriz de entrada dentro da janela de filtro, conforme mostrado na Figura 14. O agrupamento máximo é uma das operações de *pool* mais comumente usadas.

2.3.3 Camada totalmente conectada

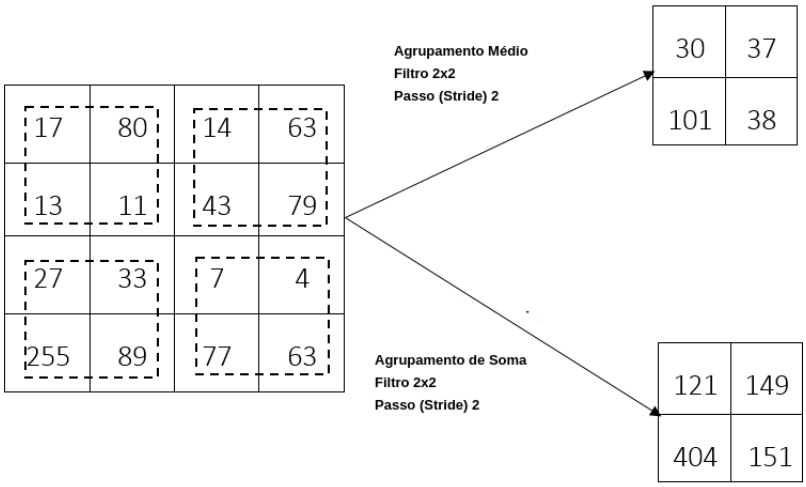
Uma *CNN* pode ter várias camadas convolucionais e camadas de *pooling*. No entanto, essas camadas apenas extrairão características da imagem de entrada e produzirão o mapa de características, ou seja, são apenas os extratores de características.

Figura 13 – Agrupamento máximo em uma matriz convolvida.



Fonte: Adaptado de Ravichandiran (2019).

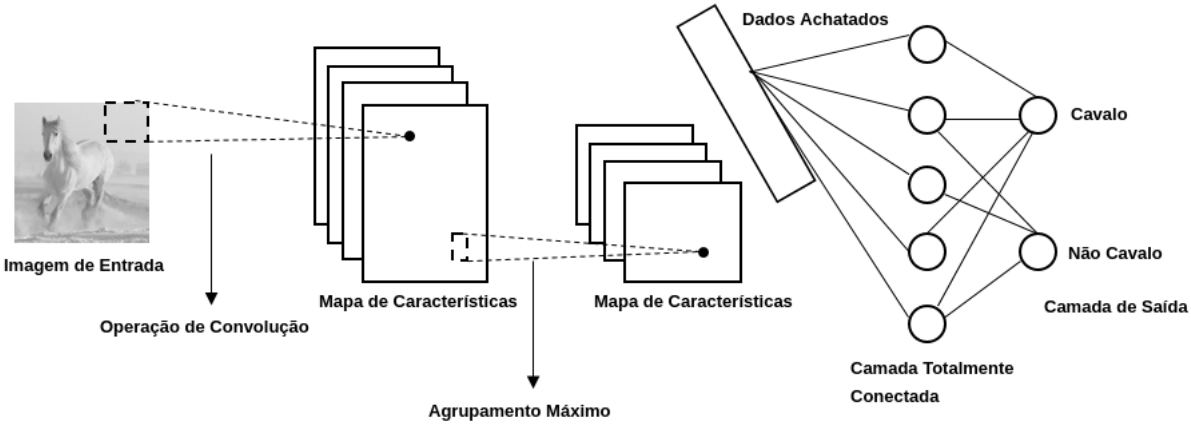
Figura 14 – Agrupamento médio e agrupamento de soma em uma matriz convolvida.



Fonte: Adaptado des Ravichandiran (2019).

Dada qualquer imagem, as camadas convolucionais extraem características da imagem e produzem um mapa de características. Posteriormente, é preciso classificar essas características extraídas. Portanto, torna-se necessário o uso de um algoritmo que possa classificar tais características e dizer se são características de um cavalo ou de outra coisa. Para realizar essa classificação, utiliza-se uma rede neural sem realimentação *feedforward*. O mapa de características é achatado e convertido em um vetor, e logo após usado como entrada para a rede *feedforward*. A rede *feedforward* obtém esse mapa de características achatado como uma entrada, aplica uma função de ativação, e retorna a saída, informando se a imagem contém um cavalo ou não. Esse processo final constitui a camada totalmente conectada e pode ser visualizado na Figura 15

Figura 15 – Camada totalmente conectada e funcionamento geral de uma *CNN*.



Fonte: Adaptado de Ravichandiran (2019).

2.3.4 Retropropagação

A operação de retropropagação se inicia após prever a saída, é calculado a perda L vista na Equação 2.8, usando o erro quadrático médio como a função de perda, ou seja, a média da diferença quadrática entre a saída real y_i, e a saída prevista $\hat{y}_i$, fornecida na Equação 2.7.

$$L = \frac{1}{2}\sum_{i}(y_i - \hat{y}_i)^2 \tag{2.8}$$

A retropropagação pode ser usada para minimizar a perda L. Primeiramente precisa-se encontrar os valores ideais para o filtro W. A matriz de filtro mostrada na Figura 9 consiste em quatro valores: $w1$, $w2$, $w3$ e $w4$ que serão utilizados como exemplo. Para encontrar a matriz de filtro ideal, precisa-se calcular os gradientes da função de perda L em relação a todos esses quatro valores. Para tal, deve-se recolher as equações da matriz de saída O utilizando a Equação 2.6.

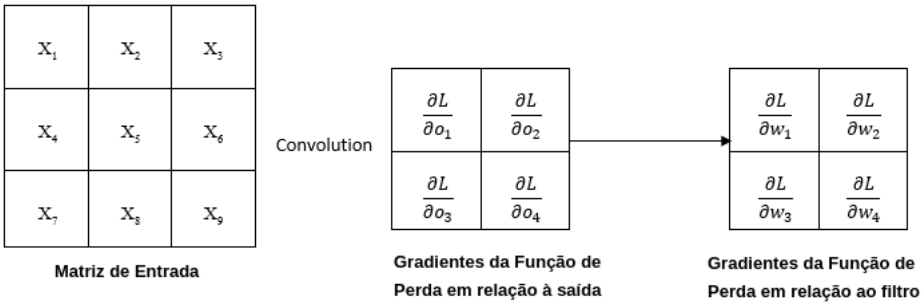
Primeiro, deve ser calculado os gradientes em relação a w_1. Como analisado nas Equações 2.9 e 2.10, w_1 aparece em todas as equações de saída, por isso calcula-se as derivadas parciais da perda em relação a w_1, onde x se refere ao *pixel* da imagem de entrada visto na Figura 9.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o_1}\cdot\frac{\partial o_1}{\partial w_1} + \frac{\partial L}{\partial o_2}\cdot\frac{\partial o_2}{\partial w_1} + \frac{\partial L}{\partial o_3}\cdot\frac{\partial o_3}{\partial w_1} + \frac{\partial L}{\partial o_4}\cdot\frac{\partial o_4}{\partial w_1} \tag{2.9}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o_1} \cdot \partial x_1 + \frac{\partial L}{\partial o_2} \cdot \partial x_2 + \frac{\partial L}{\partial o_3} \cdot \partial x_4 + \frac{\partial L}{\partial o_4} \cdot \partial x_5 \tag{2.10}$$

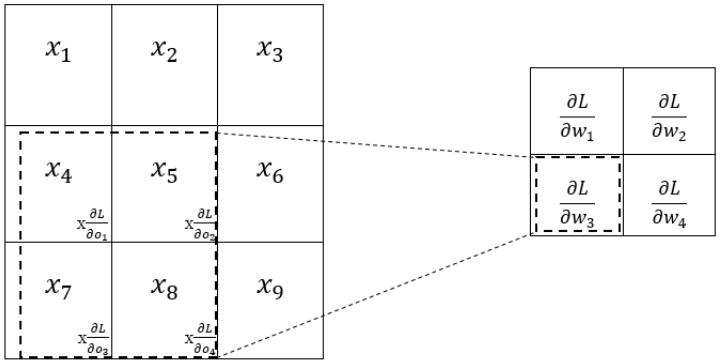
O mesmo processo é repetido para w_2, w_3 e w_4. Acontece que calcular as derivadas de perda em relação à matriz de filtro é muito simples, é apenas outra operação de convolução. Ao examinar as equações anteriores de perto, nota-se que elas se assemelham ao resultado de uma operação de convolução entre a matriz de entrada e o gradiente da função de perda em relação à saída como uma matriz de filtro, conforme ilustrado nas Figuras 16 e 17.

Figura 16 – Operação de convolução entre a matriz de entrada e o gradiente da função de perda em linha.



Fonte: Adaptado de Ravichandiran (2019).

Figura 17 – Operação de convolução entre a matriz de entrada e o gradiente da função de perda.



Fonte: Ravichandiran (2019).

Além de calcular os gradientes de perda ou gradientes da função de perda em relação ao filtro, também é necessário calcula-lo em relação a uma entrada, pois é usado para calcular os gradientes dos filtros presentes na camada anterior. A matriz de entrada usada como exemplo está presente na Figura 9 e consiste em nove valores, de $x1$ à $x9$, portanto será preciso calcular

os gradientes de perda em relação a todos eles. Da Equação 2.6 é obtido os valores de saída visto nas Equações 2.11, 2.12, 2.13 e 2.14.

$$o_1 = x_1 w_1 + x_2 w_2 + x_4 w_3 + x_5 w_4 \tag{2.11}$$

$$o_2 = x_2 w_1 + x_3 w_2 + x_5 w_3 + x_6 w_4 \tag{2.12}$$

$$o_3 = x_4 w_1 + x_5 w_2 + x_7 w_3 + x_8 w_4 \tag{2.13}$$

$$o_4 = x_5 w_1 + x_6 w_2 + x_8 w_3 + x_9 w_4 \tag{2.14}$$

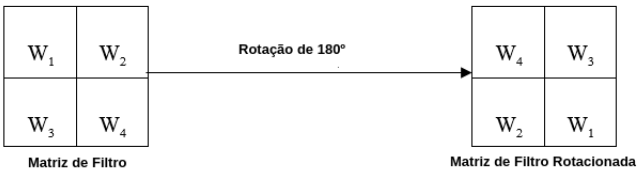
Como analisado na Equação 2.11, x_1 está presente apenas em o_1, então pode-se calcular os gradientes de perda em relação a o_1 isoladamente, anulando outros termos, como pode ser visto nas Equações 2.15 e 2.16. De uma forma muito semelhante, calcula-se os gradientes de perda em relação a todas as entradas x_i.

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial o_1} \cdot \frac{\partial o_1}{\partial x_1} \tag{2.15}$$

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial o_1} \cdot w_1 \tag{2.16}$$

Pode-se representar as equações anteriores, ou seja, os gradientes de perda em relação às entradas, usando uma operação de convolução entre a matriz de filtro como uma matriz de entrada e os gradientes de perda em relação à matriz de saída como uma matriz de filtro. Mas o truque é que, em vez de usar essa matriz de filtro diretamente, ela é rotacionada em 180 graus. Também em vez de realizar a convolução simples, realiza-se a convolução completa. Isto é feito para que seja possível derivar as equações anteriores usando uma operação de convolução. A Figura 18 mostra a aparência do *kernel* rotacionado em 180 graus.

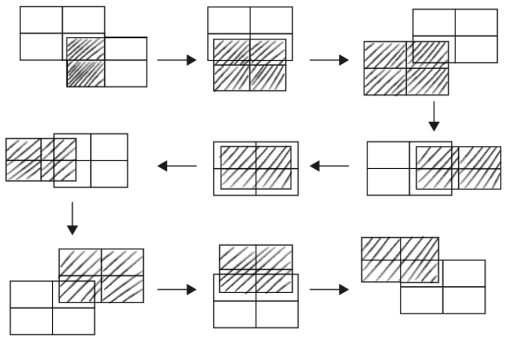
Figura 18 – matriz de filtro rotacionada em 180 graus.



Fonte: Adaptado de Ravichandiran (2019).

Da mesma forma que uma operação de convolução, em convolução completa, usa-se um filtro e que é deslizado sobre a matriz de entrada, que será deslizado de uma forma diferente da operação de convolução simples. A Figura 19 mostra como as operações de convolução

Figura 19 – Deslizamento da matriz de filtro em uma convolução completa.

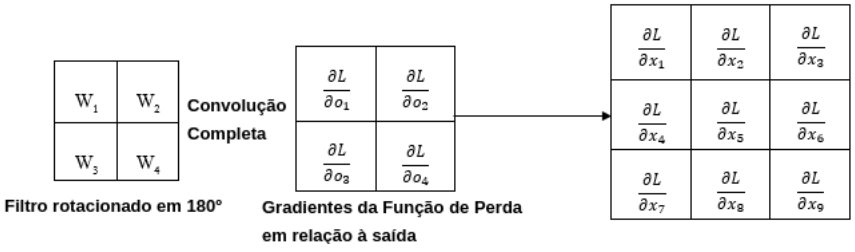


Fonte: Ravichandiran (2019).

completa funcionam. Como pode ser observado, a matriz sombreada representa a matriz de filtro e a não sombreada representa a matriz de entrada.

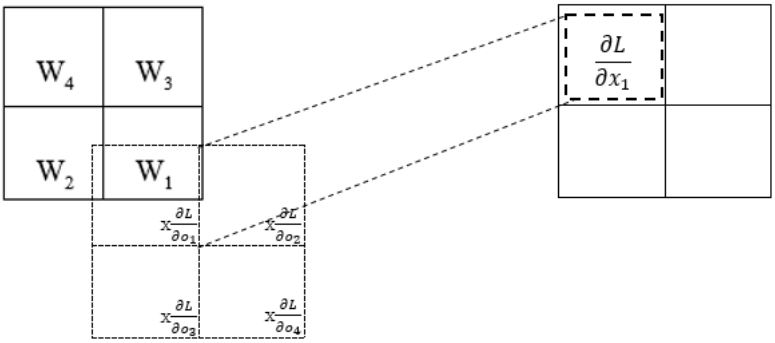
Como conclusão, o gradiente de perda em relação à matriz de entrada pode ser calculado usando uma operação de convolução completa entre um filtro rotacionado em 180 graus com a matriz de entrada e o gradiente de perda em relação à saída como uma matriz de filtro, como visto na Figura 20. Observando a Figura 21 nota-se como os gradientes de perda em relação à entrada, w_1, são calculados pela operação de convolução completa entre a matriz de filtro rotacionada em 180 graus e os gradientes de perda com em relação a uma matriz de saída como uma matriz de filtro. Também demonstrado pela Equação 2.16.

Figura 20 – Operação de convolução completa em linha.



Fonte: Adaptado de Ravichandiran (2019).

Figura 21 – Operação de convolução completa.



Fonte: Ravichandiran (2019).

3 Metodologia

Esta seção tem por objetivo apresentar os materiais e métodos utilizados na realização da presente pesquisa. Primeiramente será apresentado o modelo ou arquitetura da rede neural em questão, ou seja, a *CNN*. Posteriormente será apresentada a base de dados utilizada na rede, seguido da descrição da metodologia.

Esta pesquisa se caracteriza como aplicada, pois há aplicação utilizando uma rede neural para buscar uma solução específica. Também se caracteriza como quantitativa, pois é baseada em dados e números, precisando de técnicas estatísticas para formulação de resultados. Por fim, esta pesquisa se caracteriza também como metodológica, pois são criados métodos e instrumentos para captar informações e se chegar a determinado fim.

3.1 Arquitetura *CNN* utilizada

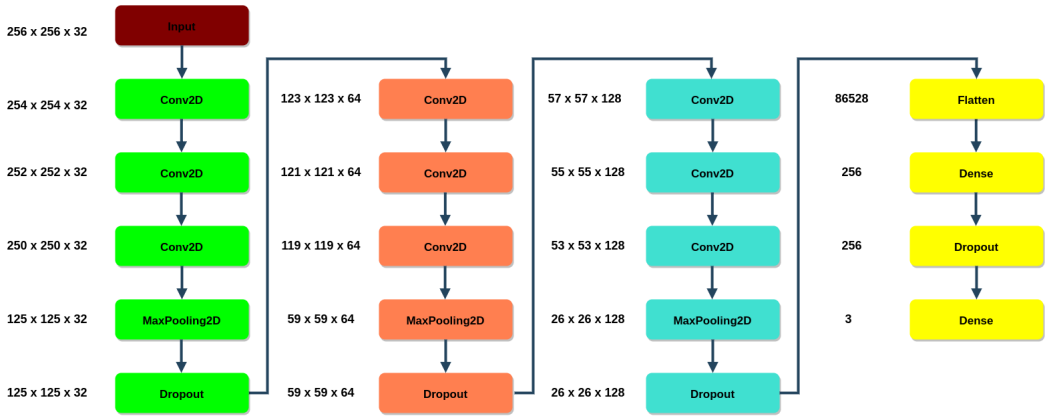
A arquitetura *CNN* utilizada neste trabalho, juntamente com as dimensões da imagem alteradas ao longo do processo de convolução, está apresentada na Figura 22. Foi utilizada uma arquitetura sequencial de três camadas idênticas com diferentes tamanhos de filtro. Todas elas possuem três módulos convolucionais (*Conv2D*) que são bancos de filtros treináveis considerados extratores de feições, seguido de um módulo de *pooling* (*MaxPooling2D*) que reduz a quantidade de recursos extraídos do módulo convolucional anterior para evitar *overfitting*, e por fim um módulo *Dropout* onde nós individuais são retirados da rede com probabilidade $1 - p$ ou mantidos com probabilidade p, de modo que uma rede reduzida é mantida. A primeira camada em verde possui um filtro de tamanho 32, a segunda em laranjado, 64, e a terceira em azul, 128. Após a terceira camada há uma sequência de quatro módulos que formam a camada de classificação: um módulo *Flatten* responsável por transformar uma matriz bidimensional de características em um vetor, um módulo *Dense* responsável por gerar uma previsão baseado nos resultados gerados pela camada de convolução, um módulo *Dropout* seguido de outro módulo *Dense*.

3.2 Base de dados

Para realização do treinamento e teste da *CNN* foi utilizada uma base de dados que possui 5150 imagens em cores brutas não compactadas, de dimensão 256×256 *pixels*, no formato *Windows Bitmap (BMP)* (LIU; COOPER; ZHOU, 2013)[1].

[1] https://www.shsu.edu/qxl005/New/Downloads/index.html

Figura 22 – Arquitetura *CNN* utilizada.



Fonte: Elaborado pelo autor (2020).

3.3 Execução do presente projeto

Como mencionado anteriormente, o objetivo principal dessa pesquisa é criar uma rede neural capaz de reconhecer fraudes em imagens, a rede em questão é multi classificadora, ou seja, além de detectar se houve fraude na imagem ou não (Inalterada - *Uncompressed*), detecta também se ocorreu *Seam Carving* (remoção de *seams*) ou *Seam Insertion* (inserção de *seams*), portanto a rede classifica em três classes de saída: *Uncompressed*, *Seam Carving* e *Seam Insertion*. O processo iniciou alterando a base de dados original citada na Seção 3.2.

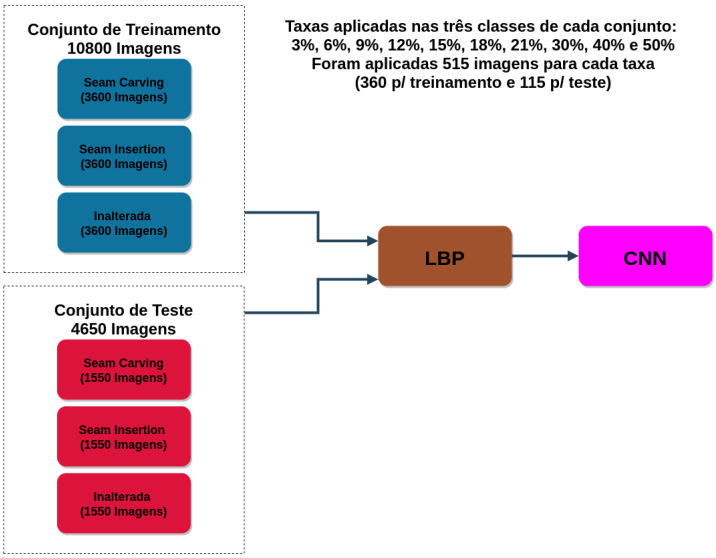
As 5150 imagens foram divididas em 10 taxas (3%, 6%, 9%, 12%, 15%, 18%, 21%, 30%, 40% e 50%) distintas de adulteração, posteriormente aplicado a cada uma dessas taxas os métodos *Seam Carving* e *Seam Insertion*. Portanto, as 5150 imagens resultaram em 10300 imagens adulteradas, logo após, divididas em 7210 imagens ou 70% para conjunto de treinamento e 3090 imagens ou 30% para conjunto de teste. Para a classe de Imagens Intocadas apenas foram selecionadas as 5150 imagens da base de dados original e divididas em 3605 imagens ou 70% para conjunto de treinamento e 1545 ou 30% para conjunto de teste. Somando todas as imagens das três classes distintas totalizou em 10815 imagens para o conjunto de treinamento e 4635 imagens para o conjunto de teste.

Como mencionado no parágrafo anterior, para aplicação dos métodos *Seam Carving* e *Seam Insertion*, o conjunto de dados foi dividido em taxas de adulteração de 3%, 6%, 9%, 12%, 15%, 18%, 21%, 30%, 40% e 50%, resultando no conjunto de treinamento em 721 imagens e no conjunto de teste em 309 imagens para cada uma das dez taxas nas duas classes distintas (CIESLAK; COSTA; PAPA, 2018)(YIN et al., 2015). A Figura 23 ilustra melhor tal divisão. Para aplicar os métodos *Seam Carving* e *Seam Insertion*, foi utilizado um algoritmo escrito em *Python*. Obtendo como exemplo a taxa de 50%, na classe *Seam Carving*, do conjunto de treinamento, as 721 imagens foram alteradas em 25% de linhas de *pixels* e 25% de colunas de

pixels em relação a imagem original, para que adulteração não ocorra só em um sentido da imagem e fique mais heterogênea.

Após todas as alterações de todas as imagens nas diferentes taxas de cada classe, em todas as imagens foi aplicado o método *LBP*, também escrito em *Python*. Por fim, a rede foi treinada e testada, os resultados são apresentados na Seção 4.

Figura 23 – Metodologia aplicada.



Fonte: Elaborado pelo autor (2020).

Todo o processo de aplicação dos algoritmos *Seam Carving*, *Seam Insertion* e *LBP*, ou seja, os algoritmos utilizados para montagem da base de dados, foi realizado manualmente pelo autor, assim como o algoritmo da *CNN*. Todos esses algoritmos foram escritos na linguagem de programação *Python* utilizando as bibliotecas *tensorflow*, *sklearn*, *matplotlib*, *numpy* e *pandas*. Todo este conteúdo pode ser encontrados no *GitHub*[2] do mesmo.

[2] https://github.com/GabesSeven/TCC_-_Trabalho_de_Conclusao_de_Curso_-_Course_Conclusion_Work

4 Resultados e Discussão

Esta seção tem por objetivo apresentar e analisar os resultados obtidos. Primeiramente é necessário entender alguns conceitos como: Época (*Epoch*), Acurácia (*Accuracy*) e Perda (*Loss*).

Em redes neurais uma Época se refere a um ciclo em todo o conjunto de dados de treinamento. Normalmente, o treinamento de uma rede neural leva mais do que algumas Épocas, ao alimentar uma rede neural com dados de treinamento por mais de uma Época em padrões diferentes, espera-se uma melhor generalização quando fornecida uma nova entrada sem rótulos (dados de teste). Uma Época costuma ser confundida com uma iteração que são os números de lote ou etapas por meio de pacotes particionados dos dados de treinamento, necessários para completar uma Época.

No momento da classificação de uma imagem no processo de treinamento, se o rótulo real for positivo e rótulo predito condiz com o rótulo real, tem-se um resultado verdadeiro-positivo VP, se não condiz ocorre um resultado falso-negativo FN. Caso o rótulo real for negativo e o rótulo predito condiz com o rótulo real, tem-se um resultado verdadeiro-negativo VN, se não condiz ocorre um resultado falso-positivo FP.

A Precisão é a forma de medir a frequência com que o algoritmo classifica um ponto de dados corretamente, também definida pela Equação 4.1, enquanto Acurácia indica a performance geral do modelo, dentre todas as classificações, quantas o modelo classificou corretamente, também definida pela Equação 4.2. Os resultados apresentados nessa seção mostra apenas as Acurácias em cada Época.

$$\frac{VP}{VP + FP} \tag{4.1}$$

$$\frac{VP + VN}{VP + VN + FP + FN} \tag{4.2}$$

Uma rede de perda é uma rede neural treinada usando um processo de otimização que requer uma função de perda para calcular o erro do modelo, sendo que esta estima o erro de um conjunto de pesos propostos na rede neural, um exemplo de função de perda é a Equação 2.8 vista anteriormente na Subseção 2.3. Em suma, a função de perda ilustra os erros de uma rede neural em um único número, de modo que quaisquer melhorias nesse número sejam indicativas de um modelo melhor.

O processo de treinamento do algoritmo de rede neural utilizado pelo autor, ou seja, o treinamento das 10815 imagens rotuladas, foi realizado utilizando vinte Épocas, sendo que os resultados obtidos podem ser visto na Tabela 1. Para cada Época, a rede divide os dados em

Tabela 1 – Resultados obtidos.

epoch	*loss*	*accuracy*	*val_loss*	*val_acc*
1	0.8617	0.4862	0.4369	0.7524
2	0.3241	0.8457	0.1644	0.9371
3	0.1542	0.9376	0.1644	0.9371
4	0.1111	0.9559	0.1018	0.9590
5	0.0887	0.9665	0.1004	0.9581
6	0.0762	0.9713	0.0441	0.9848
7	0.0668	0.9727	0.0504	0.9838
8	0.0528	0.9804	0.0281	0.9924
9	0.0515	0.9800	0.0328	0.9848
10	0.0466	0.9838	0.0126	0.9981
11	0.0366	0.9858	0.0143	0.9933
12	0.0401	0.9846	0.0352	0.9857
13	0.0319	0.9883	0.0134	0.9971
14	0.0303	0.9890	0.0119	0.9962
15	0.0215	0.9927	0.0132	0.9971
16	0.0236	0.9912	0.0085	0.9981
17	0.0222	0.9926	0.0112	0.9981
18	0.0196	0.9924	0.0045	0.9990
19	0.0168	0.9934	0.0040	0.9990
20	0.0183	0.9933	0.0106	0.9962

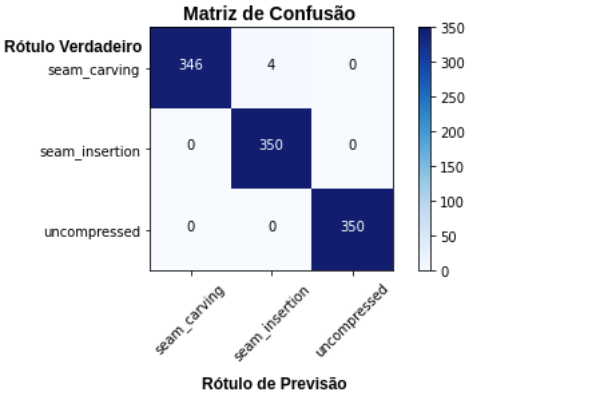
Fonte: Elaborado pelo autor (2020).

duas partes, ou seja, dados de treinamento (*loss* e *accuracy*) e dados de validação (*val_loss* e *val_acc*). Normalmente, com cada Época aumentando, a Perda (*loss*) diminui e a Acurácia (*accuracy*) aumenta. Mas com *val_loss* e *val_acc* podem ocorrer diversas variações, dentre as mais comuns: *val_loss* aumenta e *val_acc* diminui (significa que o modelo está acumulando valores, não aprendendo); *val_loss* aumenta e *val_acc* também aumenta (pode ser o caso de *overfitting*); e *val_loss* diminui e *val_acc* aumenta (correto, significa que a construção do modelo está aprendendo e funcionando bem).

Como se pode analisar na Tabela 1, praticamente todo o processo de treinamento da rede neural ocorreu de uma forma onde a Perda diminuía e a Acurácia aumentava, ou seja, o modelo funcionou corretamente nessa etapa chegando a ter uma Acurácia final muito alta, certa de 99% de acurácia.

Por fim, a Matriz de Confusão presente na Figura 24, apresenta os falsos-positivos, falsos-negativos, verdadeiros-positivos e verdadeiros-negativos. Como a rede teve 99% de acurácia, já era de se esperar poucos erros, ou seja, poucos falsos-positivos e falsos-negativos, sendo que estes só ocorreram em quatro dos exemplos como mostra a Matriz de Confusão.

Figura 24 – Matriz de Confusão obtida.



Fonte: Elaborado pelo autor (2020).

5 Conclusão

Neste trabalho foi apresentada uma técnica de detecção de fraudes em imagens, mais precisamente, foi utilizado uma *Convolutional Neural Network (CNN)* para detectar fraudes em imagens causadas por uma técnica denominada *Seam Carving*. Muitos trabalhos estão sendo produzidos envolvendo técnicas de *Deep Learning* e segurança da informação, então este é um tema importante para a área da tecnologia.

Em relação aos resultados obtidos, foram ótimos comparados aos trabalhos relacionados, pois a rede neural obteve 99% de acurácia, resultando em uma excelente distinção entre imagens adulteradas e não adulteradas. Dentre os trabalhos semelhantes, um obteve aproximadamente 98% de acurácia (CIESLAK; COSTA; PAPA, 2018), resultados muito próximos, mas provavelmente pelo modelo ou a forma com que foi manipulada a base de dados os resultados finais resultaram nessa pequena diferença, porém o trabalho dos mesmos foram mais complexos comparando os resultados com outras redes neurais padrões. Outro propôs um classificador multiclasse que possui quatro classes de saídas, 0% ou sem entalhe na costura, 10% entalhado na costura, 20% entalhado na costura e 40% entalhado na costura, o modelo de rede utilizada por esses autores obteve cerca de 84% de precisão (NAZARİ; AKGÜN, 2020). Ambos autores utilizaram uma metodologia muito semelhante em relação a rede neural utilizada e são trabalhos recentes, então é interessante tal comparação.

Um trabalho futuro poderia ser utilizar outros modelos de redes mais complexos, por exemplo uma arquitetura *Generative Adversarial Network (GAN)*, para analisar de uma forma mais ampla o método *Seam Carving*.

Referências

AVIDAN, S.; SHAMIR, A. Seam carving for content-aware image resizing. In: . New York, NY, USA: Association for Computing Machinery, 2007. (SIGGRAPH '07), p. 10–es. ISBN 9781450378369. Disponível em: <https://doi.org/10.1145/1275808.1276390>.

CHENG, H.; WEI, J.; LIN, C.; YE, J. Detecting seam-carved image by extreme learning machines using patch analysis method, jury voting, and combinatorial fusion. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, p. 1–15, 2018. ISSN 2168-2216.

CHOI, C.-H.; LEE, H.-Y.; LEE, H.-K. Estimation of color modification in digital images by cfa pattern change. *Forensic Science International*, v. 226, n. 1, p. 94 – 105, 2013. ISSN 0379-0738. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0379073812005518>.

CIESLAK, L. F. d. S.; COSTA, K. A. P. d. C.; PAPA, J. P. Seam carving detection using convolutional neural networks. In: *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. [S.l.: s.n.], 2018. p. 000195–000200.

HE, D.-C.; WANG, L. Texture unit, texture spectrum, and texture analysis. *IEEE Transactions on Geoscience and Remote Sensing*, v. 28, p. 509–512, 1990.

HUANG, D.; SHAN, C.; ARDABILIAN, M.; WANG, Y.; CHEN, L. Local binary patterns and its application to facial image analysis: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 41, n. 6, p. 765–781, Nov 2011. ISSN 1558-2442.

LIN, Y.; NIU, Y.; LIN, J.; ZHANG, H. Accumulative energy-based seam carving for image resizing. In: *17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. [S.l.: s.n.], 2016. p. 366–371.

LIU, Q. Exposing seam carving forgery under recompression attacks by hybrid large feature mining. In: *23rd International Conference on Pattern Recognition (ICPR)*. [S.l.: s.n.], 2016. p. 1041–1046.

LIU, Q.; COOPER, P. A.; ZHOU, B. An improved approach to detecting content-aware scaling-based tampering in jpeg images. In: *2013 IEEE China Summit and International Conference on Signal and Information Processing*. [S.l.: s.n.], 2013. p. 432–436.

NAZARİ, H.; AKGÜN, D. A deep learning model for image retargetting level detection. In: *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. [S.l.: s.n.], 2020. p. 1–4.

PATTERSON, J.; GIBSON, A. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2017. ISBN 1491914254.

PIETIKINEN, M.; HADID, A.; ZHAO, G.; AHONEN, T. *Computer Vision Using Local Binary Patterns*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2011. ISBN 0857297473.

RAVICHANDIRAN, S. *Hands-On Deep Learning Algorithms with Python*. Packt Publishing, 2019. ISBN 9781789344158. Disponível em: <https://books.google.com.br/books?id=MYfDwQEACAAJ>.

YE, J.; SHI, Y.; XU, G.; SHI, Y. A convolutional neural network based seam carving detection scheme for uncompressed digital images: 17th international workshop, iwdw 2018, jeju island, korea, october 22-24, 2018, proceedings. In: _____. [S.l.: s.n.], 2019. p. 3–13. ISBN 978-3-030-11388-9.

YIN, T.; YANG, G.; LI, L.; ZHANG, D.; SUN, X. Detecting seam carving based image resizing using local binary patterns. *Comput. Secur.*, Elsevier Advanced Technology Publications, GBR, v. 55, n. C, p. 130–141, nov. 2015. ISSN 0167-4048. Disponível em: <https://doi.org/10.1016/j.cose.2015.09.003>.