



Fundamentos de Visão Computacional

André Luís Resende Monteiro

2020

Fundamentos de Visão Computacional

André Luís Resende Monteiro

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

Sumário

Capítulo 1. Introdução à visão computacional	5
O que é a visão computacional.....	5
Etapas de visão computacional	8
Resumo.....	10
Capítulo 2. Aquisição e representação de imagens	11
Aquisição de imagens	11
Representação de imagens	14
Resumo.....	15
Capítulo 3. Transformações espaciais em imagens	17
Filtros espaciais	17
Histograma.....	22
Segmentação.....	26
Resumo.....	27
Capítulo 4. Descritores de imagens.....	28
Estratégia para inferência	28
Resumo.....	32
Capítulo 5. Aprendizado de máquina	33
Aprendizado de Máquina	33
Tipos de aprendizado	34
Algoritmos classificadores.....	37
Aprendizado de Máquina e Deep Learning.....	38
Motivação das Redes Neurais Artificiais	40
Estrutura básica	42
Neurônio artificial	43
Funcionamento básico	44

Algoritmo de retropropagação.....	47
Resumo.....	53
Capítulo 6. Redes Neurais Convolucionais (CNN)	56
Introdução às Redes Neurais Convolucionais (CNN)	56
Utilização do Keras	61
Pré-processamento.....	63
Camada convolucional.....	65
Camada de pooling.....	68
Camadas totalmente conectadas.....	71
Camadas de convolução transposta.....	72
Funções de ativação	74
Funções de perda	76
Resumo.....	79
Capítulo 7. Aplicações	81
Classificação.....	81
Super-resolução.....	83
Classificação + localização	83
Detecção de objetos	84
Segmentação semântica.....	85
Geração de imagens.....	87
Resumo.....	90
Referências.....	92

Capítulo 1. Introdução à visão computacional

Neste capítulo nós veremos como as soluções de visão computacional geralmente funcionam. Antes da popularização de soluções de deep learning, alguns algoritmos específicos eram utilizados comumente para realizar a tarefa de visão computacional, e estão sendo aos poucos substituídos pela utilização de redes neurais profundas.

O que é a visão computacional

Nós, seres-humanos, utilizamos nossos olhos para ver, e, mais importante, interpretar o mundo a nossa volta. Por exemplo, considere a imagem apresentada na Figura 1. Facilmente, podemos perceber um gato na imagem e, então, categorizá-la como sendo um gato.

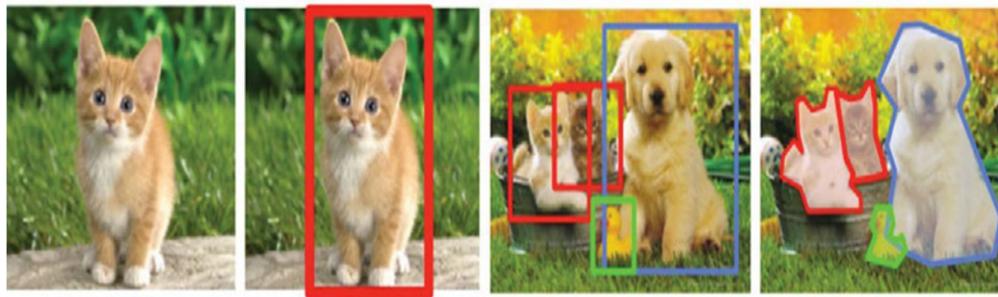
Figura 1 – Imagem de um gato.



Contudo, nós também podemos realizar outras interpretações em relação a essa mesma imagem ou imagens similares, conforme representa a Figura 2. Nós podemos, além de classificar a imagem, saber exatamente a localização do gato, por exemplo. Além disso, mesmo em uma imagem com mais de um animal (e animais diferentes), podemos encontrar exatamente onde cada um deles está na imagem e

inclusive separar cada parte da imagem, de acordo com o animal a que pertence ou se não pertence a nenhum animal (se faz parte do cenário da imagem, por exemplo).

Figura 2 – Interpretações possíveis de serem feitas pelo cérebro humano.



A visão computacional é justamente a ciência que tem como objetivo prover uma capacidade similar, ou até melhorada, aos computadores, através da criação de métodos capazes de inferir capacidades do mundo real através do processamento de imagens.

Contudo, a realização dessas tarefas de forma satisfatória não é uma atividade trivial. Diversas pesquisas já foram conduzidas com o objetivo de obter inferências a partir das imagens, considerando aspectos como bordas, variação de iluminação e cantos das imagens. Esse campo de pesquisa, contudo, permanece desafiador.

Uma distinção importante de ser feito diz respeito à diferenciação entre visão e o processamento de imagens. Geralmente, o processamento de imagens é uma tarefa anterior à visão computacional, ou seja, prepara uma ou mais de uma imagem para ser utilizada por um algoritmo de visão computacional.

Por exemplo, considere a imagem representada na Figura 3. Através dos nossos olhos, podemos perceber que se trata da mesma imagem, contudo com alguns detalhes diferentes. A imagem do lado esquerdo apresenta uma espécie de deformação, chamada de ruído, o que compromete a característica original da imagem. Apesar de várias soluções modernas de visão computacional conseguirem trabalhar com imagens desse tipo, é mais interessante utilizarmos algum tipo de

algoritmo para retirarmos esses ruídos da imagem (o lado direito da figura), antes de a utilizarmos em um processo de visão computacional. Desse modo, conseguiremos extrair informações mais precisas e de forma mais confiável da imagem.

Figura 3 – Imagem com ruído e imagem correspondente sem ruído.



Tecnicamente, o processamento de imagens utiliza como entrada uma imagem e produz como resultado uma outra imagem. No exemplo anterior, a entrada é a imagem com ruído e a saída é a imagem tratada, sem ruído. Por sua vez, a visão computacional também recebe como entrada uma imagem, mas ao invés de produzir uma outra imagem como resultado, produz alguma informação acerca dessa imagem, como por exemplo a classe a que pertence ou os objetos que a constituem.

Soluções de visão computacional podem ser aplicadas a diferentes contextos. Principalmente com a popularização de redes profundas, soluções de visão computacional estão cada vez mais populares. Soluções que usam realidade aumentada ou virtual, cada vez mais comuns em jogos e aplicações industriais, utilizam soluções de visão computacional para o seu processamento (Figura 4), assim como soluções de robótica e reconhecimento de faces.

Figura 4 – Aplicação de realidade aumentada.

Outra aplicação que ganha a cada dia mais popularidade é a pesquisa através de imagens. Devido à geração de uma grande quantidade de imagens e vídeos todos os dias, graças à, por exemplo, popularização de smartphones, é interessante que possamos descobrir imagens ou vídeos similares a uma determinada imagem que possuímos (Figura 5), recurso bastante utilizado em sites de busca.

Figura 5 – Pesquisa de similaridade entre imagens.

Etapas de visão computacional

Tradicionalmente, podemos dividir um sistema de visão computacional em algumas etapas bem definidas, cada uma com propósitos distintos. Essas etapas podem gerar tanto como saída uma outra imagem ou uma representação ou inferência a respeito da imagem. Logo, a rigor, são etapas de visão computacional ou de processamento de imagens, contudo geralmente são apresentadas juntas. É importante ressaltar também que dependendo do sistema essas etapas podem ser distintas. Inclusive, com as tecnologias mais recentes de processamento de visão

computacional, notadamente utilizando deep learning, algumas dessas etapas estão passando por profundas transformações.

Figura 6 – Etapas de um processo de visão computacional.



- **Aquisição de imagens:** Consiste na etapa de adquirir imagens do mundo real, transformando-a para uma informação capaz de ser armazenada no computador. Envolve conceitos como resolução especial, por exemplo.
- **Transformação:** Envolve manipular uma imagem e gerar outra imagem, com o objetivo de torná-la mais adequada para um propósito específico. A saída é tipicamente outra imagem, e a avaliação humana é o critério para definir se a transformação foi bem aplicada.
- **Compressão:** Envolve reduzir o tamanho da imagem, para fins de armazenamento ou transmissão, contudo sem perder a informação que o arquivo representa.
- **Segmentação:** Consiste na separação da imagem em objetos que a constituem. Essa etapa é utilizada para que se possa preparar a imagem para alguma análise automática, como por exemplo a classificação. Tradicionalmente, sempre foi um desafio em visão computacional, mas tem se tornado mais facilmente resolvível com soluções de deep learning.
- **Descrição:** Etapa que consiste em gerar uma representação para uma imagem, ou um objeto segmentado de uma imagem. Permite um

armazenamento mais fácil e rápido e idealmente capturam informações que descrevem bem a imagem.

- **Reconhecimento ou classificação:** Consiste em atribuir uma classe, ou rótulo, ao objeto descrito. Por exemplo, pode atribuir a classe “Milho bom” ou “Milho ruim” à imagem que representa um milho estragado ou com qualidade.

Resumo

Neste capítulo, compreendemos que a visão computacional consiste em obter informações de uma determinada imagem através do seu processamento. Ela é, portanto, um processo que produz informações a partir da entrada de uma imagem. Ela se difere do processamento de imagem, que também recebe como saída imagens, porém produz como saída uma outra imagem. Geralmente, o processamento de imagens é utilizado como uma fase anterior ao processo de visão computacional.

Vimos também as etapas tradicionais que compõe um sistema típico de visão computacional, e entendemos qual o propósito básico de cada uma delas. Vimos que muitas dessas etapas estão se transformando graças à popularização de tecnologias como deep learning.

Capítulo 2. Aquisição e representação de imagens

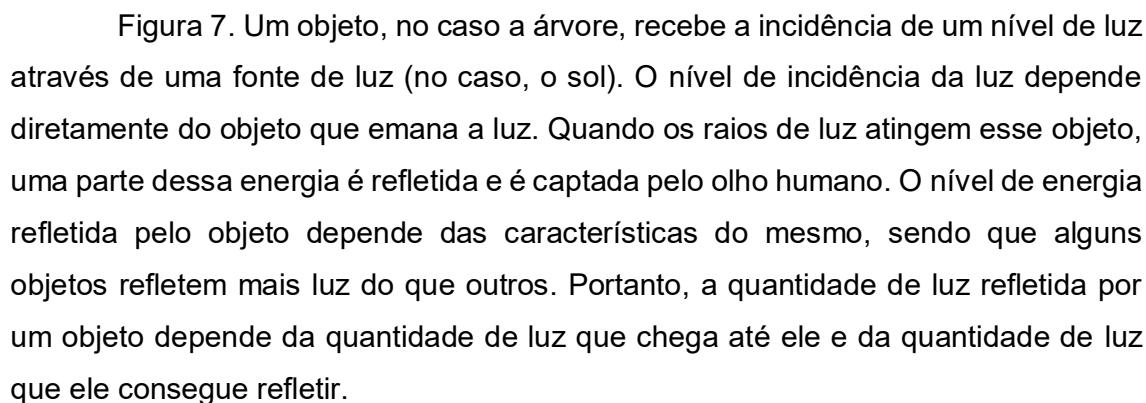
Neste capítulo veremos como acontece o processo de aquisição de imagens, ou seja, a captação de imagens do mundo real e a sua transformação para o mundo digital. Entenderemos também os principais conceitos envolvidos nesse processo.

Além disso, estudaremos como a imagem é representada no computador, com o objetivo de ser manipulada por algum software de visão computacional ou processamento de imagens.

Aquisição de imagens

A aquisição de imagens consiste na etapa do sistema de visão computacional de realizar a captação de uma imagem visualizada pelos olhos humanos, porém através de um aparelho mecânico.

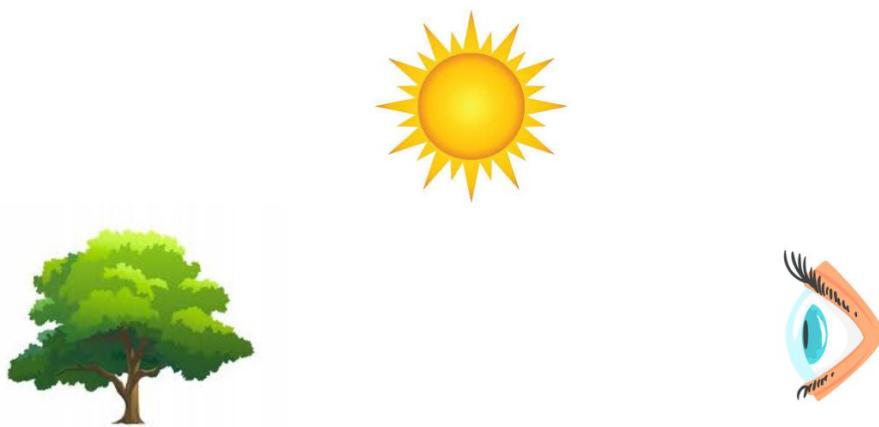
O processo de captação de imagens através do olho humano é ilustrado na



The diagram illustrates the process of light interaction with an object. It shows a scene with a sun-like source at the top left emitting rays of light. One ray of light is shown passing through the air and hitting a tree trunk in the center. The tree trunk is labeled 'Objeto' (Object). The light ray hitting the tree is labeled 'Luz incidente' (Incident light). A portion of this ray is reflected back towards the viewer's eye, labeled 'Luz refletida' (Reflected light). The eye is represented by a small circle at the bottom right, with a dashed line indicating its path towards the reflected light ray. The background is a simple landscape with a blue sky and green ground.

Figura 7. Um objeto, no caso a árvore, recebe a incidência de um nível de luz através de uma fonte de luz (no caso, o sol). O nível de incidência da luz depende diretamente do objeto que emana a luz. Quando os raios de luz atingem esse objeto, uma parte dessa energia é refletida e é captada pelo olho humano. O nível de energia refletida pelo objeto depende das características do mesmo, sendo que alguns objetos refletem mais luz do que outros. Portanto, a quantidade de luz refletida por um objeto depende da quantidade de luz que chega até ele e da quantidade de luz que ele consegue refletir.

Figura 7 – Processo de captação de imagens do olho humano.



A ideia da aquisição de imagens consiste em desenvolver objetos mecânicos que possam atuar aos moldes de um olho humano, por exemplo uma câmera. A câmera é dotada de inúmeros sensores que realizam a captação de luz refletida por objetos, aos moldes do olho humano.

Contudo, quando realizamos a captação e consequente transformação de uma imagem do mundo real para uma imagem digital, alguns aspectos devem ser definidos. Por exemplo, os níveis de detalhes da imagem e o número de cores possíveis representadas devem ser escolhidos. Uma imagem do mundo real é analógica, com detalhes e tons de cores virtualmente infinitos. Contudo, quando abrimos uma imagem digital é fácil perceber que o mesmo não ocorre. Na prática, precisamos transformar informações analógicas em informações digitais.

Figura 8 – Imagem aberta em um editor de imagens.



Considere por exemplo a Figura 8. Ela consiste em uma imagem de um carro aberta em um editor de imagens. Se ampliarmos a imagem até um certo grau, começaremos a perceber algumas irregularidades na imagem. A primeira irregularidade notada diz respeito aos detalhes da imagem, situação perceptível nas regiões de bordas. É fácil perceber que essas regiões não são contínuas, e a presentam transições bem abruptas. A segunda irregularidade diz respeito à quantidade de cores percebidas. Algumas regiões estão com as mesmas cores, diferentemente do mundo real, em que a quantidade de cores é infinita.

O processo de digitalização das coordenadas da imagem é chamado de **amostragem**, enquanto a digitalização dos valores de cor da imagem é chamada de **quantização**. Obviamente, ambos os processos produzem perdas na imagem captada. A amostragem está diretamente relacionada ao conceito de **resolução espacial**, indicando que maiores detalhes indicam uma resolução espacial mais alta. Por sua vez, a **resolução radiométrica** está relacionada ao número de cores que são representados na imagem, indicando que um número maior de cores significa uma resolução radiométrica maior. Esses conceitos são mostrados de forma visual na Figura 8 e na Figura 9.

Figura 9 – Imagens com resolução espacial menor e maior.

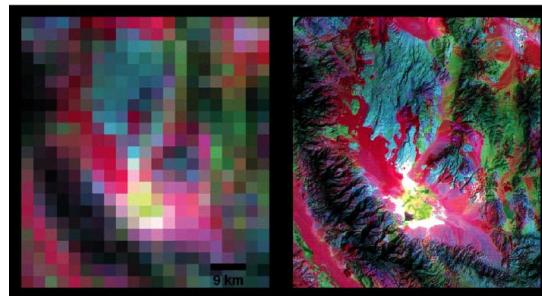
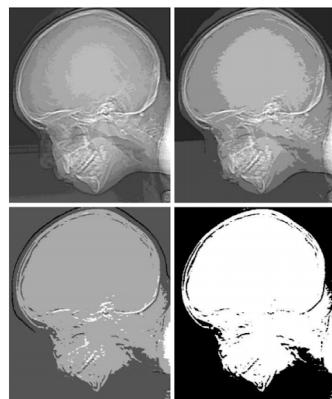


Figura 10 – Imagens com resolução radiométrica maiores e menores.



Representação de imagens

Após passar pelo processo de aquisição (por exemplo, a partir da câmera de um smartphone), a cena obtida do mundo real capturada é convertida para o mundo digital, e, portanto, armazenada e posteriormente analisada de forma numérica, conforme mostra a **Error! Reference source not found.**. Para o computador, uma imagem é uma matriz formada por um conjunto de blocos indivisíveis e contíguos,

denominados **pixels**, que possui cada um uma coordenada, representada pela tupla (x,y) , e um nível de intensidade, que representa a sua cor.

Figura 11 – Uma imagem e sua representação digital.



0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

O número máximo de pixels de uma imagem é dado pela sua resolução espacial, e o número de intensidades distintas possíveis para cada pixel é dada pela resolução radiométrica, ambos dependendo do tipo de aquisição realizada. Logo, o tamanho que uma imagem ocupa no disco é diretamente proporcional a essas duas grandezas. A fórmula que calcula o tamanho da imagem é dada por $\text{pixels} * 2^{RR}$, em que RR represe a resolução radiométrica da imagem. Uma resolução radiométrica de 1 indica que a imagem armazena apenas duas cores (tipicamente, preto e branco).

O valor de intensidade de cada pixel varia entre 0 e 255 para uma imagem apenas em tom de cinza, ou seja, sem cores, sendo que o 0 representa o preto, o 255 representa o branco e um valor intermediário entre 0 e 255 representa um nível de cinza intermediário. Para representar imagens coloridas, um pixel passa a possuir mais de um valor. Esse conjunto de valores, e a forma como eles são representados, depende do **sistema de cor** utilizado.

Existem diversos tipos de cores que podem ser utilizados, como o CMYK, o HSL e o HSV, cada um com um propósito específico. Contudo, o sistema de cores mais utilizados em computadores é o chamado RGB, sigla em inglês para *Red, Green and Blue*. Nesse sistema de cor, cada pixel possui três valores, que variam entre 0 e 255, indicando o nível de vermelho, verde e azul que possui. O 0 representa a

ausência de cor, e o 255, a presença total da cor. Dessa forma, um pixel com o valor (255, 0, 0) no sistema RGB representa o vermelho puro, visto que suas demais componentes (verde e azul) são 0. Uma ferramenta possível para visualizar os valores de pixel em RGB pode ser obtido em https://www.w3schools.com/colors/colors_picker.asp. A representação em RGB ocupa três vezes mais espaço que uma representação em tom de cinza, visto que cada pixel possui três valores.

Resumo

Neste capítulo, vimos que a aquisição de imagens é responsável por capturar informações do mundo real através de um equipamento mecânico, e realizar a sua digitalização através dos conceitos de amostragem e quantização. Associados a eles também existem os conceitos de resolução espacial e resolução radiométrica.

A imagem é representada digitalizada através de uma matriz, em que cada elemento é denominado pixel, que possui uma coordenada e um valor de intensidade. Geralmente, o valor de um pixel é representado entre 0 e 255, sendo que 0 representa o preto e 255, o branco. Para imagens coloridas, um sistema de cores é utilizado. O RGB é o sistema de cores mais utilizado em computadores. No RGB, cada pixel possui três valores, indicando os níveis de vermelho, verde e azul.

Capítulo 3. Transformações espaciais em imagens

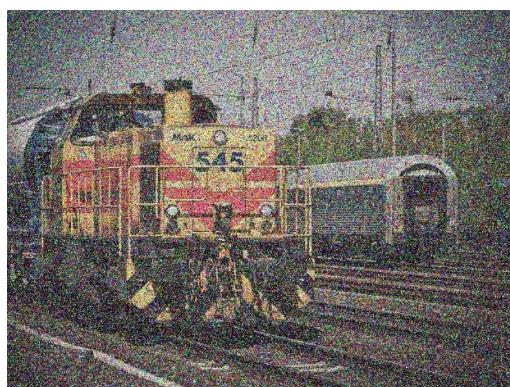
Neste capítulo, apresentaremos as principais manipulações que podemos realizar através da manipulação de imagens, que tem como objetivo alterar o valor dos seus pixels para atingir algum objetivo específico.

Filtros espaciais

Uma vez que a imagem é armazenada digitalmente, pode ser necessário realizar alguma transformação sobre ela, com o objetivo de se atingir algum objetivo específico, como a remoção de interferências na imagem ou o destaque de alguma parte em especial. As transformações que são realizadas diretamente sobre os valores dos pixels da imagem são denominadas transformações espaciais.

Transformações são realizadas por exemplo para a remoção de **ruídos**. Ruídos são variações na intensidade dos pixels que não eram esperadas, e geradas principalmente através de interferências no processo de aquisição da imagem. A Figura 12 representa uma imagem digital representada com alguns ruídos. Perceba que existem anomalias na imagem, o que compromete a sua visualização.

Figura 12 – Ruído em uma imagem digital.



Uma forma de se realizar as transformações espaciais é através dos filtros espaciais. Os filtros espaciais são matrizes que executam o processo de convolução matemática sobre uma determinada região da imagem. Essas matrizes são

posicionadas inicialmente no início da imagem, e, a cada processo de convolução, são deslizadas um pixel para o lado, aos moldes de uma janela deslizante que percorre toda a imagem. Ao final do processo, uma nova imagem é gerada, com os seus valores de pixels alterados. Para se aplicar um filtro a uma imagem RGB, aplique-se o filtro aos 3 canais da imagem, e depois junta-os novamente.

Os valores que compõe cada filtro produzem resultados diferentes nas imagens. Especialmente, os filtros espaciais são utilizados para dois propósitos distintos: a suavização e o realce de bordas.

Para se entender o funcionamento específico dos filtros, é necessário se entender os conceitos de frequências de pixels. Regiões da imagem que possuem muitos pixels iguais são denominadas regiões de baixa frequência; por outro lado, regiões da imagem com valores diferentes são denominadas regiões de alta frequência. Logo, a frequência é relativa à variação de intensidade dos pixels que compõe determinada região da imagem.

Considere, por exemplo, a Figura 13. As regiões que possuem muitos pixels com valores parecidos, como o céu, nuvem, mar ou mesmo a cor azul do carro são consideradas regiões de baixa frequência na imagem. Por sua vez, regiões que possuem muitos detalhes, como as rodas, faróis ou maçaneta do veículo são consideradas regiões de alta frequência, em que há grande variação nos valores dos pixels do carro. Geralmente, regiões de alta frequência representam bordas entre objetos, pois nesses locais há o predomínio de valores diferentes de intensidade.

Figura 13 – Imagem com regiões de baixa e alta frequências.



Os filtros de suavização, como o próprio nome diz, têm como objetivo suavizar a imagem original. Também são conhecidos como filtros **passa-baixa**, pois permitem que baixas frequências sejam mantidas na imagem, enquanto regiões de alta frequência (essencialmente bordas) sejam descartadas. São particularmente utilizados para a remoção de ruídos, pois eles se tratam de altas frequências na imagem. Contudo podem comprometer o detalhe das imagens, gerando o efeito de borramento.

A Figura 14 representa a aplicação de um filtro de borramento em uma imagem, em que se variou o tamanho do filtro. Perceba que, à medida que o tamanho do filtro aumentou, os efeitos de borramento ficaram mais realçados, e algumas regiões de alta frequência na imagem (detalhes da letra, contornos dos círculos) não puderam mais ser observados.

Figura 14 – Aplicação de um filtro de suavização do tipo média.



Um tipo de filtro possível de ser aplicado é o filtro da média. Nesse caso, o valor do pixel resultante na imagem transformada será igual à média de todos os valores presentes dentro da região da matriz de convolução. Uma variante do filtro média é a média ponderada, em que se pode favorecer determinados pixels em detrimento dos demais. Essa análise deve ser feita empiricamente dependendo do problema que se esteja resolvendo.

As matrizes de convolução da média e da média ponderada são exibidas na Figura 15.

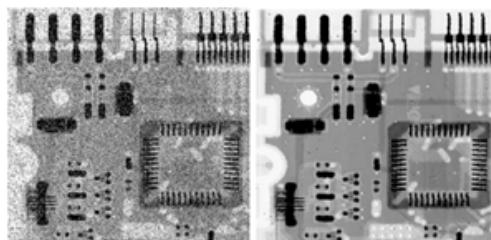
Figura 15 – Matrizes de convolução da média e da média ponderada.

$\frac{1}{9} \times$	<table border="1"><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	1	1	1	1	1	$\frac{1}{16} \times$	<table border="1"><tr><td>1</td><td>2</td><td>1</td></tr><tr><td>2</td><td>4</td><td>2</td></tr><tr><td>1</td><td>2</td><td>1</td></tr></table>	1	2	1	2	4	2	1	2	1
1	1	1																			
1	1	1																			
1	1	1																			
1	2	1																			
2	4	2																			
1	2	1																			

Outro tipo de filtro são os filtros não-lineares. Nesses filtros, o processo de convolução não é realizado. Ao invés disso, o valor na matriz gerada será igual a uma função qualquer não-linear aplicada aos pixels que se encontram dentro da janela deslizante. Um exemplo de filtro não-linear é o filtro da mediana. Nesse filtro, os pixels da janela são ordenados e a mediana desse conjunto é colocada na imagem de resultado. Esse filtro é particularmente aplicado para ruídos do tipo sal e pimenta, pois esse ruído gera pixels aleatórios com valores muito próximos entre 0 e 255.

A Figura 16 mostra uma imagem com esse tipo de ruído em que foi aplicado o filtro da mediana.

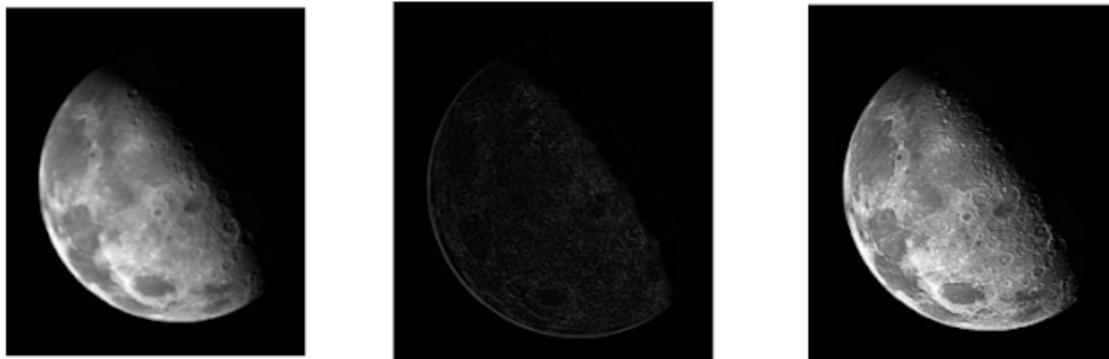
Figura 16 – Imagem com ruído sal e pimenta e aplicação do filtro mediana.



Outros filtros são os filtros de realce de bordas, conhecidos também como filtros **passa-alta**, pois permitem que regiões da imagem com altas frequências (geralmente bordas) sejam passadas para a imagem resultante, ao passo em que elimina regiões com baixa frequência. A utilização desse filtro envolve algumas

etapas: primeiramente, eles são aplicados à imagem original, produzindo apenas as suas bordas; logo em seguida, essa imagem das bordas é somada à imagem original (soma normal dos pixels), gerando uma outra imagem com as bordas destacadas. Esse processo é exibido na Figura 17.

Figura 17 – Exemplo de aplicação de um filtro passa-alta.



Exemplos de filtros passa-alta são o laplaciano, em que suas variações são exibidas na **Error! Reference source not found.**. Os filtros de Prewitt (Figura 19) e o Sobel (Figura 20) também realizam a captura de bordas, possuindo também a variação para capturar inclusive apenas bordas horizontais ou verticais, dependendo do que se pretende ressaltar.

Figura 18 – Exemplos de filtros laplacianos,

0	1	0	1	1	1
1	-4	1	1	-8	1
0	1	0	1	1	1
0	-1	0	-1	-1	-1
-1	4	-1	-1	8	-1
0	-1	0	-1	-1	-1

Figura 19 – Filtro Prewitt para obtenção de bordas horizontais e verticais.

-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	0	1

Figura 20 – Filtro Sobel para obtenção de bordas horizontais e verticais.

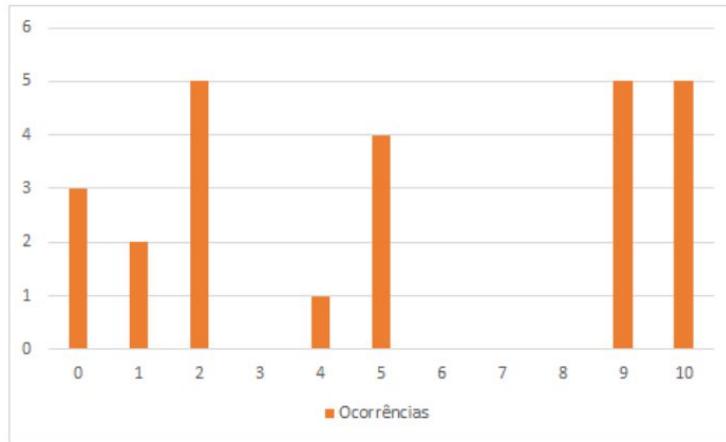
-1	-2	-1	-1	0	1
0	0	0	-2	0	2
1	2	1	-1	0	1

Histograma

O histograma de uma imagem digital pode ser definido como uma função que indica quantos pixels na imagem possuem cada valor de intensidade de cor, entre 0 e 255. Considere por exemplo a **Error! Reference source not found.**, que representa a representação de uma imagem fictícia à esquerda e o seu respectivo histograma à direita. Observe, que por exemplo, o número de ocorrências do valor de intensidade 10 é igual a 5, o que é representado no seu histograma. Esse processo é feito para todos os pixels da imagem (assume-se que os valores de 11 a 255 não possuem valor, por isso o seu resultado é 0 e não são exibidos na imagem). O histograma normalizado é obtido dividindo-se o valor de cada ocorrência pelo total de pixels, de forma que o seu valor fique normalizado entre 0 e 1.

Figura 21 – Histograma de uma imagem fictícia.

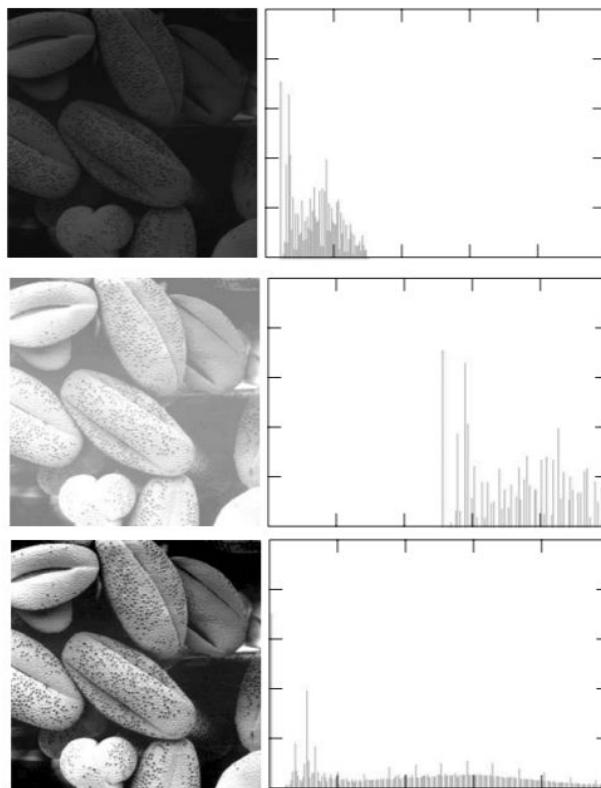
0	0	0	1	1
2	2	2	2	2
10	10	10	10	10
9	9	9	9	9
5	5	5	5	4



O histograma de uma imagem é uma ferramenta útil que nos ajuda a entender como os valores de intensidade dos pixels estão distribuídos dentro de uma imagem. Considere, por exemplo, a **Error! Reference source not found.**. Ela representa uma mesma imagem original manipulada três vezes, de formas distintas. Perceba que a concentração de nível de intensidade em cada uma delas é facilmente percebida nos seus histogramas. A primeira variação representa uma imagem mais escurecida, em que os valores do seu histograma estão mais próximos do 0 (ou seja, preto). A segunda variação, uma imagem mais clara, em que o seu histograma está mais concentrado em valores próximos a 255. A terceira, uma imagem em que os valores de intensidades estão mais bem distribuídos ao redor da imagem.

Essa maior nitidez da terceira imagem é devido ao fato de ela apresentar um **contraste** maior. O contraste pode ser definido como a diferença das intensidades de um pixel dentro de uma imagem. Imagens que possuem valores mais distribuídos de intensidades possuem um contraste maior.

Figura 22 – Três histogramas de uma mesma imagem original.



Uma forma de se ajustar o contraste de uma imagem é através da técnica conhecida como equalização do histograma. A técnica consiste em se construir a chamada função de distribuição acumulada (fda) de cada intensidade presente na imagem. Após isso, normaliza-se esses valores entre 0 e 255. Esse processo é representado na

Figura 23 para a imagem fictícia representada na **Error! Reference source not found.**. Perceba que o valor da fda para cada nível de intensidade é calculado através da soma do seu número de ocorrência mais todas as ocorrências nos níveis anteriores. A equalização de cada nível de intensidade do histograma é dado por arredondamento($\frac{fda(v)-fda_{min}}{(M \times N) - fda_{min}} \times (L - 1)$), em que $fda(v)$ é o valor da fda de cada

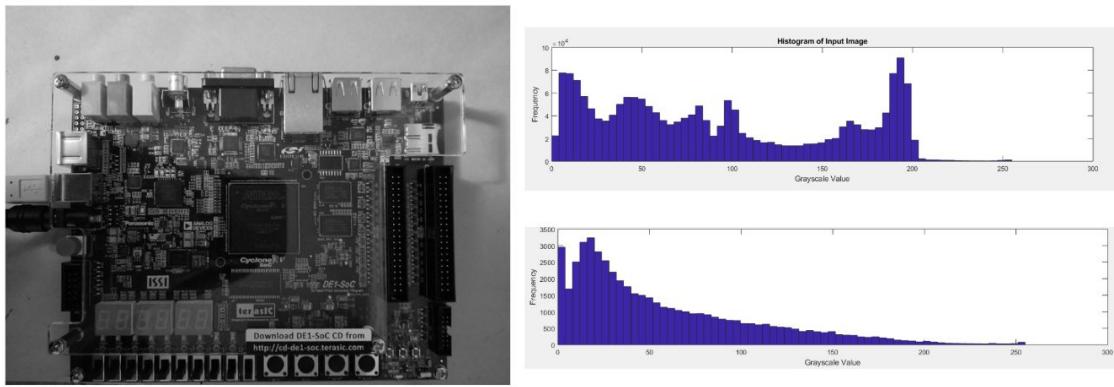
linha da tabela, fda_{min} é o menor fda da tabela, $M \times N$ é o número de pixels da imagem e L é o número de intensidades da imagem (geralmente, 256).

Figura 23 - Equalização do histograma da imagem fictícia anterior.

Valor	Fda	Equalização
0	3	0
1	5	23
2	10	81
4	11	92
5	15	139
9	20	197
10	25	255

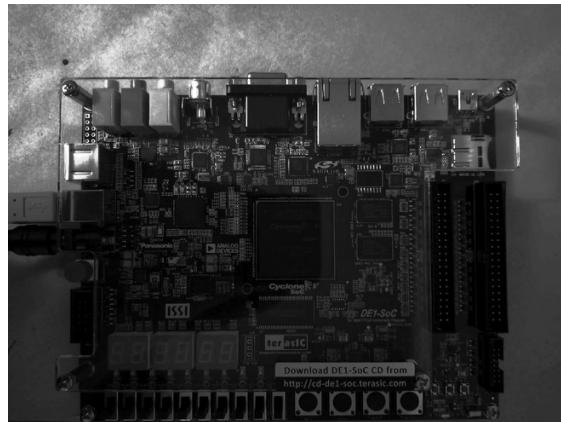
Uma outra técnica aplicada ao histograma é a conhecida como casamento do histograma. Nessa técnica, faz-se com que o histograma de uma imagem tende a se parecer com um outro histograma conhecido, com algum propósito específico. Por exemplo, considere a Figura 24, que representa uma imagem, seu histograma original (histograma superior) e um histograma em que se quer realizar o casamento da imagem (histograma inferior). É possível perceber que o histograma inferior produzirá uma imagem mais escurecida, visto que suas intensidades estão mais armazenadas em valores mais próximos ao 0.

Figura 24 – Imagem com dois histogramas para casamento.



O resultado do casamento desse histograma é mostrado na Figura 25.

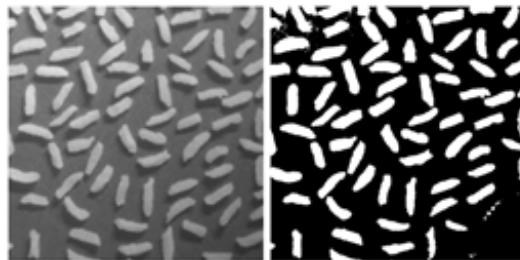
Figura 25 – Resultado da aplicação do casamento do histograma.



Segmentação

O processo de segmentação de imagens tem como objetivo separar os objetos ou áreas que compõe uma imagem, a partir do seu entendimento semântico dentro da imagem. Por exemplo, a Figura 26 mostra esse processo executado em uma imagem obtida através de um microscópio. A imagem à direita mostra as células individuais da imagem segmentadas, cada uma representada por objetos individuais coloridos de branco.

Figura 26 – Resultado do processo de segmentação de imagens.



Geralmente, a segmentação é utilizada como preparação para algum processo de inferência sobre a imagem, como por exemplo avaliar se uma determinada peça possui defeito. Tradicionalmente, a segmentação de imagens é uma tarefa complexa de ser realizada, mas adventos na área de deep learning a tem tornado cada vez mais factível.

As técnicas tradicionais de segmentação de imagens podem ser divididas em duas categorias: baseadas em descontinuidade ou baseadas em similaridade. No primeiro grupo, o objetivo é primeiramente realizar a identificação de linhas e bordas na imagem, e após estas serem encontradas, as áreas da imagem são definidas a partir das suas fronteiras.

As técnicas baseadas em similaridade são algoritmos que iterativamente agrupam pixels vizinhos baseados em algum critério de similaridade entre eles. Um exemplo dessa técnica é a técnica conhecida como crescimento de regiões. Ela começa a partir da escolha de um pixel denominado como semente. A partir desse pixel, o algoritmo avalia se os pixels vizinhos possuem um nível de similaridade de cor, por exemplo, em relação ao pixel semente. Caso tenham, eles são então agrupados como pertencentes ao mesmo grupo.

Resumo

Neste capítulo, vimos como podemos realizar transformações espaciais em imagens, que são aquelas em que se altera diretamente os valores dos pixels. Uma forma de se realizar as transformações espaciais é através dos filtros espaciais, que são matrizes que percorrem a imagem original, promovendo transformações nos seus

pixels à medida que a vai processando. Os filtros podem ser utilizados especialmente para suavização ou realce de bordas.

Outra forma de se realizar a manipulação de imagens é através do seu histograma, em que podemos, por exemplo, aumentar o contraste da imagem. Uma outra etapa de transformação, utilizada previamente a alguma inferência, é a etapa de segmentação. O seu objeto é separar a imagem nos objetos ou nas áreas que a compõe, baseadas em detecção de bordas ou em algum critério de similaridade.

Capítulo 4. Descritores de imagens

Neste capítulo estudaremos uma outra etapa muito importante nos métodos tradicionais de visão computacional. Trata-se dos descritores de imagens que possuem como objetivo criar uma representação alternativa da imagem para que alguma inferência possa ser realizada.

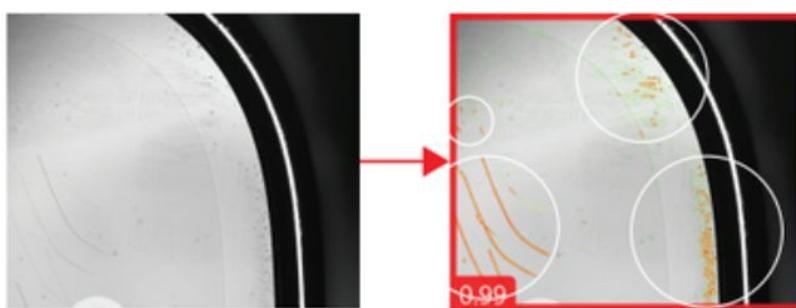
Estratégia para inferência

Tipicamente, existem duas formas para realizarmos a inferência de alguma informação em algum processo de visão computacional: a inferência manual ou a inferência automatizada.

Inferência manual

Considere, por exemplo, a tarefa de se criar um sistema de visão computacional para avaliar se um disco que é produzido possui alguma irregularidade ou deformidade, como por exemplo o disco representado na **Error! Reference source not found.**. Uma estratégia possível é a construção de um algoritmo específico para esse problema que, após realizar a aquisição da imagem, utiliza as transformações espaciais estudadas nos capítulos passados para tomar alguma decisão sobre o problema. Esse algoritmo é um conjunto de instruções, aos moldes de um programa de computador tradicional.

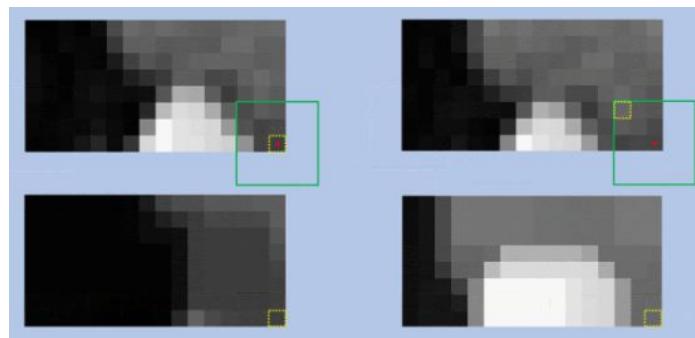
Figura 27 – Processo de inspeção de deformidade em um disco.



Além das transformações estudadas, outras podem ser utilizadas. Nesse problema específico, é possível por exemplo utilizar a técnica conhecida como limiarização (threshold), que, a partir da criação do histograma da imagem, verifica se ela possui algum tom de cor que destoa dos demais e que pode representar um defeito. Nesse exemplo, a limiarização pode procurar por regiões mais escuras da imagem, e então destacá-las, transformando a sua cor. Muitas das vezes, contudo, é necessário saber exatamente a cor ou intensidade que determinado defeito possui na imagem capturada.

Outras técnicas que podem ser utilizadas são a **erosão** e a **dilatação**, representadas na Figura 28. No processo de erosão, as regiões mais claras da imagem são contraídas, enquanto na dilatação as regiões mais claras são expandidas. Dependendo do problema que estejamos resolvendo, podemos utilizá-las para destacar uma outra região da imagem.

Figura 28 – Processos de erosão e dilatação em imagens.



O *template matching* também é uma técnica bastante utilizada. Ela consiste em, dada uma imagem de referência, procurar essa mesma imagem dentro da imagem que está sendo processada. Isso é especialmente útil em linhas de inspeção industrial, por exemplo, em que se conhece o formato da peça com a qual se está lidando, podendo, portanto, criar um template fixo com o qual se tenta “casar” a imagem processada. Contudo, essa estratégia é falha quando os formatos dos objetos analisados são muito distintos, ou quando não se consegue manter uma distância fixa entre a câmera e o objeto.

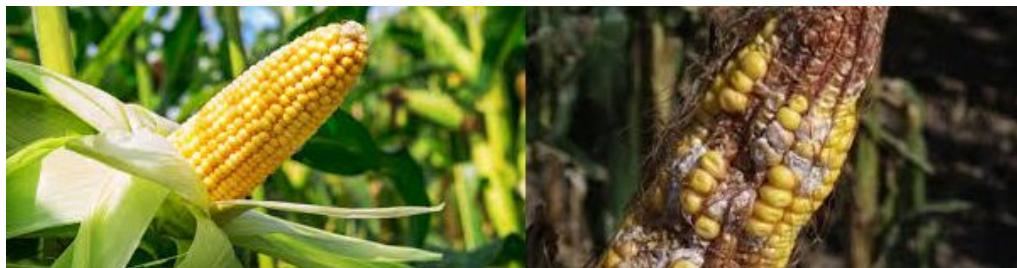
Estratégias de inferência manual podem funcionar bem em várias situações, porém é necessário que uma solução específica seja desenvolvida para cada problema, o que torna o processo menos reutilizável. Além disso, é necessário que se tenha um conhecimento muito grande naquele problema, já que a solução irá lidar com aspectos bem particulares da imagem com a qual se está trabalhando.

Inferência automatizada

A partir da geração cada vez maior de imagens, em diversos contextos, como na internet ou através de celulares, novas estratégias mais automáticas para realizar tarefas de visão computacional começaram a ser propostas. A inferência automática foi pensada inicialmente para realizar a tarefa de classificação, que consiste em atribuir uma classe, ou rótulo, a uma imagem.

Considere por exemplo a tarefa de distinguir espigas de milhos estragadas de espigas de milhos boas para o consumo, em uma indústria de produção de milhos. O ideal é que se tenha um processo automático que, dada a imagem, consiga dizer se ela se refere a um milho estragado ou um milho bom, conforme mostra a **Error! Reference source not found.** Esse processo é ainda mais complexo pois as espigas de milho não são regulares quanto uma peça mecânica. Existem muitas variações entre diversas espigas, mesmo dentro da mesma classe, o que pode fazer com que a inferência manual seja pouco aplicável.

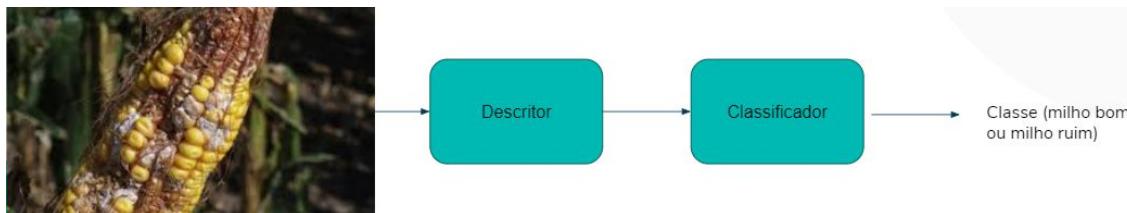
Figura 29 – Duas espigas de milho, uma sadia e outra doente.



Uma estratégia para lidar com problemas como esse é a combinação de dois tipos de algoritmos específicos, denominados descritores e classificadores. Nessa abordagem, a imagem a ser analisada passa pelo algoritmo descritor, que cria uma

representação alternativa para ela. Essa representação é então enviada para um algoritmo classificador, que realiza a sua classificação entre uma das classes disponíveis. Esse processo é ilustrado na Figura 30.

Figura 30 – Processo de classificação automática de imagens.



A função do algoritmo descritor é criar uma representação que defina bem a imagem, excluindo informações desnecessárias, como regiões do fundo da imagem e de baixa frequência, que não possuem muita significância. Além disso, a dimensão da descrição gerada por esse algoritmo é menor que a dimensão original da imagem, o que permite com que o algoritmo de classificação opere de forma mais eficiente.

Os algoritmos descritores operam buscando extrair informações que discriminem bem a imagem, como bordas, cantos, traços e variação de cores. Uma boa descrição deve ser invariante a transformações na imagem, como escala e rotação, visto que essas alterações não mudam o objeto representado na imagem. Exemplos de algoritmos descritores são o SIFT (1999), Hog (2005) e Surf (2006).

Uma lacuna dos algoritmos de descrição é que eles são genéricos, ou seja, funcionam basicamente da mesma forma para todos os tipos de imagens. Isso significa que as mesmas informações são capturadas em problemas de classificação de milhos, classificação de espécies de animais ou inspeção de uma linha de motor, por exemplo. Não é difícil, perceber, contudo, que esses problemas possuem características distintas, que podem fazer com informações mais específicas precisem ser capturadas para o problema seja mais bem resolvido.

Resumo

Nesse capítulo vimos que estratégias de inferência manual são algoritmos que utilizam códigos explicitamente programadas a partir das características da imagem para que algum processo de inferência seja realizado. As transformações espaciais estudadas até aqui podem ser utilizadas para esse fim.

Contudo, para problemas mais complexos e que possuem maior variabilidade, novas estratégias, mais automatizadas, foram propostas. Uma dessas estratégias é a utilização de algoritmos descritores, que criam uma representação menor da imagem, capturando aspectos que melhor a discriminam. No próximo capítulo, veremos como os algoritmos classificadores são utilizados para realizar a inferência automática da classificação.

Capítulo 5. Aprendizado de máquina

Neste capítulo veremos o que é o aprendizado de máquina, como ele se relacionada com o deep learning e algumas das motivações pelas quais estudá-lo.

Aprendizado de Máquina

Para resolver qualquer problema computacional, é necessário o desenvolvimento de um algoritmo, que tem como objetivo receber um conjunto de dados de entrada e produzir uma determinada saída. Por exemplo, para ordenar um valor de números, podemos construir um algoritmo que receba como entrada um vetor aleatório e produza como saída o mesmo vetor com os seus elementos ordenados.

Considere que queiramos construir um algoritmo que categorize um e-mail em duas categorias: spam ou não-spam. Ou seja, dada uma entrada de uma mensagem de e-mail, a saída produzida deverá ser a categorização daquela mensagem em sendo spam ou não sendo spam. Para um problema como esse, não é factível (ou mesmo possível) a criação de um algoritmo eficiente, uma vez que a definição de spam varia ao longo do tempo e é particular para cada usuário.

Contudo, podemos utilizar dados anteriores para solucionar esse problema. É possível consolidar milhares de mensagens que se saiba previamente que se tratam ou não de spam, com o objetivo de ajudar na tomada de decisão da categorização. O que se procura nesse caso é um processo que se aprenda com os dados anteriores e que consiga, através desses exemplos, definir o que categoriza uma mensagem como sendo spam. Em outras palavras, o que se espera é que a máquina desenvolva automaticamente um algoritmo para desempenhar essa tarefa.

Com os avanços em hardware e a popularização de mecanismos de armazenamento, os dados gerados todos os dias possuem um volume gigantesco, podendo ser acessados de diversas localidades geográficas através da Internet. Contudo, transformar esses dados em informações úteis para um negócio, de forma

estratégica, continua sendo um desafio e objeto de estudo em diversas áreas da ciência da computação.

Considere um outro exemplo: uma loja virtual, que comercializa centenas de itens por dia. A cada compra, os bancos de dados da empresa armazenam dados diversos, tais como a localidade e idade do comprador, a hora da transação e o caminho que o cliente fez até realizar a compra. Isso tipicamente gera um volume de vários *gigabytes* todos os meses. Enquanto esses dados estão apenas armazenados nos servidores da empresa, não possuem muita relevância, a não ser por exemplo contabilizar as compras e calcular a lucratividade do negócio. Contudo, será possível o desenvolvimento de um algoritmo que, baseado nesses dados anteriores, predizer quais os consumidores mais possíveis para um determinado item?

Novamente, não há um algoritmo de fácil construção para esse problema. Assim como no exemplo das mensagens de spam, a solução varia com a localidade, com o tempo ou até mesmo com o tipo de produto comercializado. Procura-se nesse caso, portanto, uma solução que consiga automaticamente gerar um algoritmo que possa predizer corretamente (ou com a maior aproximação possível), os clientes mais aptos a comprar determinado tipo de item. A partir desse ponto, os dados da organização passam a ser mais bem aproveitados e tornam-se muito mais valiosos para a empresa.

No exemplo anterior, talvez não seja possível identificar as relações entre os itens e consumidores de forma perfeita. Contudo, o objetivo do aprendizado de máquina é a construção de modelos computacionais, a partir da identificação de padrões, que tenham uma aproximação boa e útil. O que é uma aproximação “boa” e “útil” irá variar dependendo do problema que iremos solucionar.

Tipos de aprendizado

As tarefas que uma solução de aprendizado de máquina visa executar podem ser classificadas de acordo com a natureza do aprendizado desempenhado.

Aprendizado supervisionado

Em aprendizados do tipo supervisionado, há a presença de um conjunto de dados, ou amostras, como entrada, e, para cada uma das entradas, também se conhece a sua saída.

Considere novamente o problema de classificação de e-mails entre spam ou não-spam. Em uma solução de aprendizado do tipo supervisionada, vários e-mails previamente categorizados entre spam ou não-spam estariam disponíveis. Ou seja, para cada entrada (e-mail), saber-se-ia previamente a sua saída correspondente (spam ou não-spam).

O objetivo do aprendizado supervisionado é encontrar uma forma de relacionar os dados de entrada (e-mails) com as suas respectivas saídas (spam ou não-spam), de forma que, quando uma nova amostra é obtida, possa-se predizer a sua saída através da relação encontrada. A saída de cada amostra é conhecida como *label*, rótulo ou classe. Esse processo de aprendizagem funciona obtendo-se relações que mapeiem as entradas em saídas e verificando a sua corretude através da comparação com os dados previamente disponíveis.

Problemas em que só se possui a presença de duas classes (spam ou não-spam, por exemplo), são tradicionalmente conhecidos como problemas binários. Contudo, várias classes podem existir em um cenário. Por exemplo, considere a tarefa de se classificar os dados de um cachorro em uma raça específica, através de dados como peso, altura, idade, tamanho do focinho etc., a partir de um conjunto de amostras pré-classificadas. Esse problema pode possuir várias classes possíveis, cada uma correspondendo a uma raça de cachorro presente no cenário analisado.

Aprendizado não-supervisionado

Em soluções de aprendizado do tipo não-supervisionado, há a presença de um conjunto de amostras de entrada, porém não se conhece as suas respectivas saídas, ou classes. O objetivo nesses casos é encontrar regularidades ou relações entre as entradas, através da identificação de padrões que ocorrem mais em relação

a outros. Como não se possui a presença das classes relativas a cada amostra, não é possível avaliar a solução em termos de acurácia.

Um método muito utilizado de treinamento não-supervisionado é o que se conhece como *clustering* ou agrupamento, em que o objetivo é agrupar os elementos em um conjunto de grupos, a partir da relação e semelhança entre eles.

Considere novamente o exemplo de uma companhia que comercializa produtos on-line. Uma solução de agrupamento poderia ser utilizada para agrupar os clientes da companhia em determinados perfis de clientes, através da semelhança entre os seus hábitos de consumo e comportamento no site. Dessa forma, o setor de *marketing* da empresa poderia adotar estratégias de publicidade específicas para cada grupo, de forma a otimizar os recursos gastos e otimizar a conversão dos clientes.

Aprendizado semi-supervisionado

Em soluções de aprendizado do tipo semi-supervisionado há a presença de um grande conjunto de amostras, porém apenas algumas são previamente mapeadas. É um meio termo entre os aprendizados supervisionado e não-supervisionado.

No mundo real, não são todos os problemas que se possui a facilidade de se obter amostras previamente mapeadas, como o exemplo de classificação de e-mails como spam ou não-spam. Muitas das vezes obter essas amostras pode ser difícil em termos de tempo ou custo.

Considere por exemplo o problema de se classificar uma imagem de um sapo entre as diversas espécies de sapo existentes. Mesmo obtendo-se um conjunto grande de imagens de sapos (entrada), a classificação de cada imagem na sua respectiva classe (saída) demanda a presença de um especialista no domínio, o que torna o processo custoso em termos financeiros.

Vários problemas de aprendizado de máquina acabam lidando com esse tipo de situação. Uma situação possível pode ser a utilização de soluções denominadas

de *active learning*, ou aprendizado ativo, em que se busca selecionar as melhores amostras para serem categorizadas por um especialista no domínio, evitando a necessidade de todas as amostras serem mapeadas.

Algoritmos classificadores

Algoritmos de classificação são algoritmos supervisionados de aprendizado de máquina, utilizados para analisar e reconhecer padrões. Na nossa tarefa de visão computacional de distinguir milhos bons de milhos ruins, os algoritmos de classificação recebem um conjunto de descrições geradas a partir de um grande conjunto de imagens e extrai as características presentes no vetor de descrição que melhor representa cada uma das classes.

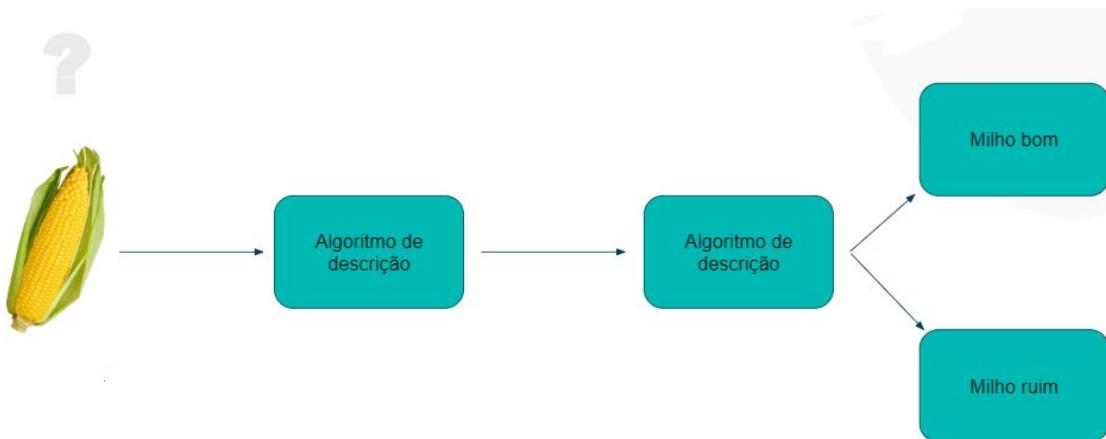
Para que esse algoritmo consiga operar bem, é necessário um grande conjunto de dados para que ele aprenda a definir os padrões. Esse conjunto de dados é chamado **conjunto de treinamento**. Cada elemento do conjunto de treinamento deve estar previamente classificado como pertencente a uma das classes. Logo, é necessário que um especialista no domínio do problema realize essa classificação prévia. Durante o processo de treinamento, o algoritmo classificador analisa todas as imagens do conjunto de treinamento e aprende as características que melhor definem as classes.

A partir daí, quanto uma nova imagem, para a qual não se sabe a classe deve ser avaliada, o processo opera conforme a Figura 31. A imagem entra no algoritmo descriptor utilizado durante o treinamento, que então gera então a sua descrição. Essa descrição serve de entrada para o algoritmo de classificação que, a partir das inferências que já possuía, realiza a sua classificação entre milho bom ou milho ruim.

O sucesso de um algoritmo de classificação é dependente da quantidade e da qualidade dos dados de treinamento. Quanto mais dados de qualidade o conjunto de treinamento possui, melhor o algoritmo consegue obter relações que realizam a distinção entre as classes. Exemplos de algoritmos de classificação são a Máquina

de vetores de suporte (SVM), as árvores de decisão, o Random Forest, o k-NN e a Naive Bayes.

Figura 31 – Avaliação de uma nova imagem com descrição e classificação.



Aprendizado de Máquina e Deep Learning

O deep learning ou aprendizagem profunda, é parte de um conjunto de métodos de aprendizado de máquina, que tem, dentre outros propósitos, oferecer uma alternativa às soluções de aprendizado de máquina construídas de forma manual.

O deep learning tem a sua origem nas Redes Neurais Artificiais (RNAs), a serem discutidas no próximo capítulo, que tiveram a utilização aumentada de forma acentuada com a popularização das unidades de processamento gráficas (GPUs), e a da disponibilidade de um grande conjunto de dados através da Internet. Várias soluções que utilizam deep learning já são conhecidas, em diversos setores.

É importante ressaltar que deep learning também é um tipo de aprendizado de máquina. Contudo, para facilitar o entendimento ao decorrer deste texto, chamaremos de “aprendizado de máquina” aquelas soluções tradicionais de aprendizado, em detrimento ao deep learning.

Uma das diferenças principais entre soluções de deep learning e algoritmos tradicionais de aprendizado de máquina quando aplicados à visão computacional, é que os classificadores utilizam descritores genéricos, que operam da mesma forma sobre todas as imagens. Contudo, soluções de deep learning possuem a característica de definir de forma automática as características específicas que melhor discriminam um tipo de problema, podendo capturar informações que os descritores tradicionais não conseguiam.

Deep learning também funciona como descritor e classificador ao mesmo tempo, funcionando de uma forma integrada. Isso significa que o resultado da classificação pode influenciar a forma com o que o descritor é construído, de forma dinâmica, sempre se adaptando às características específicas do problema que se esteja resolvendo.

Outra característica de soluções baseadas em deep learning é que elas extraem as características de forma hierárquica. Por exemplo, uma solução desse tipo primeiro poderia identificar bordas que diferenciam os dois animais, agrupar essas bordas para formar estruturas mais complexas e agregar todas essas informações para obter a resposta final.

Se compararmos soluções tradicionais de aprendizado de máquina com deep learning veremos que soluções tradicionais necessitam de menos dados históricos para operar bem, porém a performance do deep learning com um conjunto grande de dados é maior. Em relação a hardware, deep learning necessita de maiores recursos para funcionar bem, devido ao seu funcionamento interno, que iremos abordar nos próximos capítulos.

Soluções tradicionais de deep learning necessitam de um especialista no domínio do problema para definir manualmente as características que serão utilizadas para realizar o processamento. Contudo, em alguns problemas (por exemplo, diferenciar várias espécies de sapos ou identificar tumores em uma imagem médica), o especialista pode não estar disponível ou ser muito oneroso a sua utilização.

Por fim, soluções baseadas em deep learning levam mais tempo para serem construídas (o que será abordado mais a frente como fase de treinamento). Além disso, apresentam a característica de serem fim-a-fim, ou seja, muitas vezes não é trivial entender as características que foram utilizadas para realizar a computação, já que muitas vezes só são significativas pelo computador. A inteligibilidade da solução, portanto, é menor que em soluções tradicionais.

Motivação das Redes Neurais Artificiais

As pesquisas com RNAs iniciaram-se através da constatação de que o cérebro humano era particularmente bom para determinar tarefas específicas particulares, como por exemplo o reconhecimento de padrões. Por exemplo, considere a tarefa de identificação de dígitos escritos à mão. Apesar de o mesmo dígito, escrito por pessoas diferentes, tenha características distintas, como espessura e angulação, conforme exibido na Figura 32, o cérebro humano consegue de forma relativamente fácil distingui-los.

Figura 32 - Exemplos de dígitos escritos à mão.



Contudo, o mesmo não acontece com o computador digital convencional, que processa informações de uma forma diferente. Por exemplo, para os computadores é fácil o processamento de multiplicações de números grandes, como 24×72 , o que não é uma tarefa fácil para o cérebro humano.

Diante disso, a motivação para criar uma estrutura computacional que remetesse ao funcionamento do cérebro humano teve ao menos duas motivações principais:

1) Entendimento detalhado do cérebro humano: A partir da criação de modelos computacionais que simulassem o funcionamento do cérebro humano, seria possível entender melhor o seu próprio funcionamento, o que propiciaria avanços em pesquisas de diversas áreas, como a medicina, biologia e psiquiatria.

2) Automatização de tarefas: Através da criação de modelos computacionais que simulassem o funcionamento do cérebro humano, tarefas fáceis de serem executadas por humanos, como o reconhecimento de dígitos, poderiam ser realizadas também por computadores. Essa é uma motivação inerente da área da ciência da computação.

Dessa forma, a primeira citação ao termo neuro computação remete a 1943, em artigos que sugeriam a criação de uma máquina inspirada na estrutura do cérebro humano. De fato, de forma geral uma rede neural artificial é uma máquina que é projetada para simular a forma com que o cérebro humano desempenha uma tarefa específica.

Avanços na ideia de uma máquina que simulasse o funcionamento do cérebro humano foram acontecendo, até que em 1975 a primeira rede neural artificial com várias camadas foi proposta. Contudo, devido à limitação de processamento disponível na época, as redes neurais levavam semanas ou meses para terminarem o seu processamento, o que tornava a sua utilização inviável.

A partir do ano de 2009, devido sobretudo à popularização de processamento e facilidade de obtenção de GPUs (que conseguem paralelizar diversos cálculos executados pelas redes), as RNAs começaram a ser referência na área de inteligência artificial, começando a vencer diversas competições de reconhecimento de padrões e aprendizado de máquina.

Estrutura básica

Uma RNA tem como objetivo funcionar de forma parecida com o funcionamento do cérebro humano. O cérebro humano consiste em um conjunto bastante denso de células nervosas interconectadas, responsáveis por processar informações, denominadas neurônios. De acordo com Shepherd (2003), o cérebro humano possui aproximadamente 10 milhões de neurônios e 60 trilhões de conexões entre eles, denominadas sinapses.

Os sinais elétricos são propagados de um neurônio para os outros através de reações eletroquímicas pelas sinapses, e suas intensidades podem ser alteradas, por exemplo, dependendo dos padrões que estejam sendo analisados. Isso leva ao fato de que a intensidade das conexões entre os neurônios varia, e inclusive novas conexões entre neurônios distintos podem ser formadas.

O cérebro humano pode ser considerado um grande sistema, complexo, não-linear e que realiza processamento paralelo. As informações são armazenadas e processadas simultaneamente por toda a rede neural cerebral. Além disso, possui a capacidade de aprender através das experiências, alterando a sua constituição de sinapses e neurônios. O processo de aprendizado é uma tarefa fundamental no cérebro humano, razão pela qual não é difícil concluir que o seu funcionamento atraiu interessados em estudá-lo para o desenvolvimento de soluções de aprendizado de máquina.

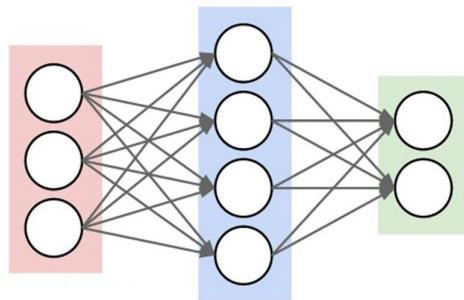
Apesar de a aproximação das RNAs com o cérebro humano ser muito primitiva, seus resultados são animadores. As redes neurais artificiais, assim como a rede neural cerebral, é capaz de aprender com experiências passadas e generalizar o seu aprendizado a outras amostras que ainda não haviam sido encontradas.

A estrutura básica de uma RNA é exibida na Figura 33. Ela consiste em um conjunto de unidades de processamento, chamadas neurônios, e diversas ligações, responsáveis por transmitir informações de um neurônio a outro, a exemplo do que ocorre no cérebro humano. Cada ligação entre um par de neurônios tipicamente tem um peso, novamente inspirado no que acontece nas sinapses cerebrais.

Cada neurônio pode receber uma ou mais informações de entrada de outros neurônios, porém produz apenas uma saída, apesar de ela poder ser conectada a vários outros neurônios. Cada neurônio nas RNAs é organizado no que se conhece como **camadas**.

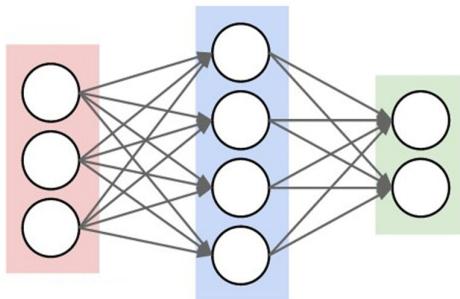
Uma camada é exibida como um conjunto de neurônios organizados na vertical. No exemplo da figura, existem três camadas: a primeira (rosa) é denominada camada de entrada, e é onde as informações começam a trafegar. A última camada (verde) é conhecida como camada de saída, e onde se localizam os neurônios que irão dar a resposta da rede. As camadas localizadas entre as de entrada e saída são denominadas camadas ocultas (ou hidden), e é onde as informações são processadas até que sejam enviados à camada de saída.

Figura 33 – Estrutura básica de uma RNA.



Neurônio artificial

A estrutura de um neurônio artificial é detalhada na Figura 34. Todos os neurônios da rede, com exceção dos neurônios da camada de entrada, recebem como entradas informações oriundas de outros neurônios ($x_1, x_2 \dots x_n$) cada uma delas associada a um peso diferente ($w_1, w_2 \dots w_n$).

Figura 34 – Estrutura básica de uma RNA.

O neurônio então agrupa todos esses resultados através do somatório do valor de cada entrada multiplicado pelo seu respectivo peso. Um outro parâmetro, chamado **bias**, é somado ao valor final, e tem como objetivo funcionar ao modo de uma constante em uma função linear. O valor final do cálculo do neurônio é então dado por $\sum x_i w_i + b$, em que b é o valor do bias.

Contudo, antes de propagar o valor calculado para os próximos neurônios, o resultado passa por uma função denominada **função de ativação**. O seu objetivo é transformar a função linear presente nos neurônios em uma função não-linear, o que permite à rede modelar cenários cada vez mais complexos. Caso não fosse utilizada uma função de ativação não-linear, toda a RNA poderia ser resumida a uma única função linear.

Funcionamento básico

Abordaremos agora o funcionamento genérico de uma rede neural artificial. Considere por exemplo a tarefa de se reconhecer um dígito presente em uma imagem escrita à mão. De uma forma simplificada e ingênuas (veremos uma forma mais otimizada adiante), cada pixel da imagem poderia ser lido e associado a um neurônio da camada de entrada. Essas informações então seriam propagadas aos neurônios das camadas ocultas, que executariam operações matemáticas sobre eles. Os resultados dessas operações seriam transmitidos aos neurônios da camada de saída, que agregariam o resultado e dariam uma resposta, indicando a qual dígito aquela

imagem se refere. Esse tipo de rede em que não há ciclos, ou seja, a informação sempre percorre uma mesma direção, são chamadas de redes do tipo **feedforward**.

O mecanismo de aprendizado de uma rede neural feedforward pode ser comparado por analogia ao mecanismo de aprendizado humano. Inicialmente, os pesos das ligações entre cada neurônio e seu bias podem ser iniciados de forma aleatória, quando ainda não se sabe sobre o domínio do problema que se está analisando. Para que a rede consiga ajustar os seus pesos em um aprendizado supervisionado, são necessárias amostras pré-classificadas, que formam o que se conhece como **conjunto de treinamento**. No exemplo dos dígitos escritos à mão, um conjunto de treinamento seria formado por várias imagens de dígitos escritos à mão, previamente categorizadas (0, 1, 2, 3 ... 9).

A cada iteração da rede, as imagens do conjunto de treinamento são processadas, entrando pelos neurônios da camada de entrada, passando pelas camadas ocultas e chegando até à camada de saída. O resultado obtido no neurônio da camada de saída é então comparado com o resultado previamente categorizado daquela amostra (no caso do problema de dígitos escritos à mão, haveriam 10 saídas possíveis no neurônio da camada de saída, correspondendo aos dígitos de 0 a 9).

A função que calcula a diferença entre o resultado obtido pela rede para uma determinada amostra e o seu resultado previamente categorizado é chamado de **função de perda**. A função de perda é calculada para cada amostra no conjunto de treinamento. O ideal em uma rede é que a função de perda sempre se iguale a zero. Dessa forma, a predição que a rede estaria fazendo e a categorização real do objeto seriam iguais. Uma função de perda comumente utilizada é a função de perda 0-1, cuja fórmula é exibida na Figura 35.

Figura 35 – Função de perda 0-1.

$$L = \begin{cases} 1, & \text{se } |a - a'| \geq \theta \\ 0, & \text{se } |a - a'| < \theta \end{cases}$$

Nessa função, o valor previamente categorizado da amostra (a) é subtraído do valor encontrado pela rede (a'). Caso o resultado seja maior que um limiar (Θ), a função assume o valor 1; caso contrário, assume o valor 0 (ou seja, não há perda).

Obviamente, a função 0-1 apenas retorna os valores 0 ou 1, indicando se houve ou não perda. Contudo, outras funções que retornam valores intermediários podem ser utilizadas, dependendo da aplicação que esteja sendo desenvolvida.

A função que agrupa os resultados individuais das funções de perda obtidas para amostra é denominada **função de custo**, e indica a perda total que se teve em relação às amostras do conjunto de treinamento. Uma função utilizada é a função **erro quadrático médio**, exibido na figura 36. Essa função realiza a média das funções de custo de cada amostra (x_i) elevadas ao quadrado.

Figura 36 – Função erro médio quadrático.

$$C = \frac{1}{N} \sum_{i=1}^N L(x_i)^2$$

Após a função de custo ser calculada, o comportamento padrão da rede neural envolve a atualização dos pesos de cada neurônio, com o objetivo que a função de custo seja minimizada. Esse processo é denominado **treinamento da rede**. Determinar quanto os pesos vão ser alterados depende do resultado da função de custo (se foi alta ou baixa) e de um outro parâmetro definido na rede: a **taxa de aprendizado**.

A taxa de aprendizado essencialmente diz respeito à velocidade com que a rede realiza o seu aprendizado. Contudo, nem sempre definir uma taxa de aprendizado alta é uma boa escolha. Suponha por exemplo que na primeira passada a rede tenha categorizado corretamente a primeira metade das amostras de dígitos escritos à mão, porém tenha errado a segunda metade delas. Se a taxa de aprendizado for muito alta, os pesos serão alterados de forma a categorizar corretamente a segunda metade das amostras, porém pode vir a errar a primeira metade, e nunca chegar a convergir a um resultado satisfatório.

Contudo, caso a taxa de aprendizado seja baixa demais, a rede pode demorar muito a terminar a sua execução. A definição desse parâmetro é uma solução de compromisso que deve ser feita, procurando um valor que seja baixo o bastante para que a rede consiga convergir a um resultado útil, mas alto o bastante para que ela não leve meses ou mesmo anos para terminar de ser treinada.

Geralmente o treinamento da rede termina em duas situações: quando o resultado da função custo atingir um valor considerável, ou que significa que a rede alcançou uma alta taxa de acerto; o quando um número determinado de passadas por todas as amostras de treinamento tenha sido alcançada. Cada passada pela rede por todas as amostras de treinamento recebe o nome de **época**.

Algoritmo de retropropagação

O algoritmo de retropropagação é o algoritmo mais utilizado para realizar a atualização dos parâmetros que formam uma rede neural de múltiplas camadas. Não entraremos em detalhes matemáticos neste material, pois essencialmente o que nos interessa é a intuição por trás do algoritmo e o seu funcionamento básico. Contudo, aqueles que quiserem realizar um aprofundamento no seu funcionamento, podem consultar as referências disponibilizadas neste material, especialmente em Hecht-Nielsen (1992).

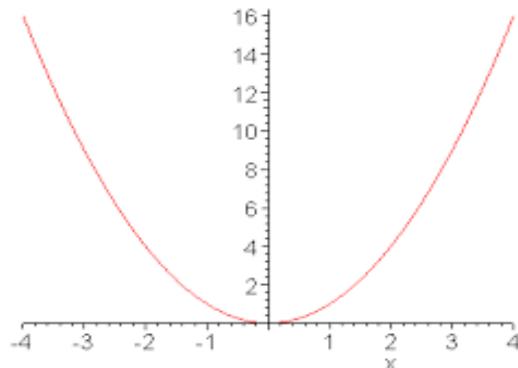
A versão moderna do algoritmo de retropropagação foi proposta em 1970, por Seppo Linnainmaa, um estudante de mestrado finlandês. Ele passou por diversas melhorias durante os anos, mas sua essência continua a mesma.

A motivação para a utilização do algoritmo é simples: como todos os parâmetros (pesos e bias) de uma rede neural podem ser atualizados da forma mais efetiva e computacionalmente viável, de forma que a acurácia (ou taxa de acerto) da RNA continue aumentando? A solução mais simples possível para isso seria utilizar a força bruta: poderíamos gerar todas as combinações possíveis de parâmetros e testá-las. Contudo, para a nossa rede de reconhecimento de dígitos escritos à mão,

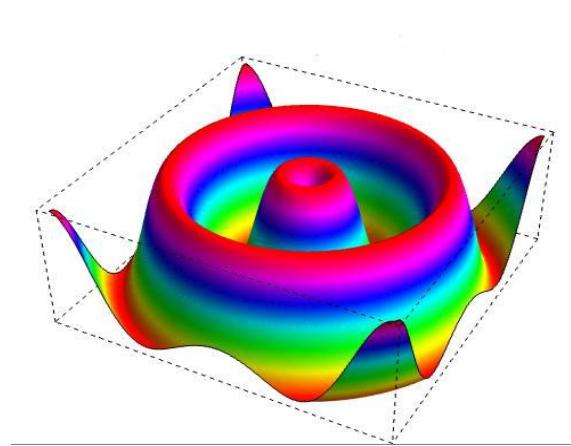
com 13.002 parâmetros, isso produziria um total de combinações que deixaria a RNA testando durante séculos!

Uma solução melhor é utilizar o conceito matemático de vetor gradiente. O vetor gradiente é uma ferramenta matemática que indica a direção na qual obtém-se o crescimento máximo do valor de uma função qualquer. Por exemplo, considere a função de uma parábola exibida na Figura 37. Nesse gráfico, considere um ponto $X = 2$, com seu respectivo valor $Y = 4$. Existem duas formas possíveis de se alterar o valor de Y : aumentando ou reduzindo o valor de X . Uma dessas alterações (aumentar ou reduzir X) irá provocar o maior aumento no valor de Y . Obviamente, nesse caso, ao se aumentar o valor de X , o valor de Y também aumenta. O cálculo do vetor gradiente nessa situação iria indicar justamente um valor positivo, ou seja, teríamos que aumentar o valor de X para que Y também aumentasse, assim como foi possível constatar em uma simples checagem visual.

Figura 37 – Função representando uma parábola.



Contudo, nem sempre é possível fazer essa checagem de forma visual. Considere agora a função em três dimensões exibida na Figura 38. Não é possível, saber visualmente com certeza qual a direção que maior maximiza o resultado da função (ou seja, aumentar ou diminuir os valores de X e Y), independente do ponto em que se esteja localizado. Todavia, através do cálculo do vetor gradiente, essa constatação poderia ser feita de forma relativamente simples.

Figura 38 – Função em três dimensões complexa.

O mesmo raciocínio pode ser aplicado para a função de custo utilizada na rede neural artificial. Considerando novamente a RNA utilizada para reconhecer dígitos a mão com 10 neurônios na camada de saída (um representando o quanto a imagem corresponde a cada dígito), caso uma imagem de um dígito 2 entre na rede, idealmente o que espera são os seguintes resultados na camada de saída:

Neurônio Dígito 0: 0,0

Neurônio Dígito 1: 0,0

Neurônio Dígito 2: 1,0

Neurônio Dígito 3: 0,0

Neurônio Dígito 4: 0,0

Neurônio Dígito 5: 0,0

Neurônio Dígito 6: 0,0

Neurônio Dígito 7: 0,0

Neurônio Dígito 8: 0,0

Neurônio Dígito 9: 0,0

Ou seja, nessa situação a RNA teria categorizado essa imagem da forma esperada. O único neurônio ativado haveria sido o que indica que o dígito 2 foi encontrado, enquanto todos os outros estariam zerados. Utilizando como função de custo a função de erro médio quadrático, poderíamos calcular o quanto a rede errou em categorizar essa imagem, através da fórmula presente na Figura 39, em que Y_{di} representa o resultado desejado como saída no neurônio i ($0, 0, 1, 0, 0, 0, 0, 0, 0, 0$) e Y_i representa o resultado obtido. Quanto maior o valor da função de custo, pior foi o resultado da rede na categorização daquela amostra.

Figura 39 – Função erro médio quadrático utilizada como função de custo da rede.

$$C = \frac{1}{10} \sum_{i=1}^{10} (Y_{di} - Y_i)^2$$

O valor de Y_i pode ser reescrito essencialmente em função dos pesos e bias dos neurônios anteriores, já que é assim que ele é calculado. Dessa forma, a função de custo descrita poderia ser reescrita utilizando apenas pesos, bias e funções de ativação. O importante a se notar é que essa função de custo é uma função qualquer, como as duas anteriormente exibidas. Portanto, pode-se calcular o vetor gradiente da forma tradicional e obter-se a direção em que a função de custo iria aumentar à medida em que alterasse os seus parâmetros (os pesos e bias).

Contudo, como o nosso objetivo é reduzir a função de custo (de forma que se aproxime de 0, e que a saída da rede seja igual à resposta encontrada), calcula-se o oposto do vetor gradiente, multiplicando-o por -1. Ou seja, o vetor obtido indicaria em que direção devemos alterar os pesos e bias de toda a rede para que caminhemos em direção a um erro 0. Devido a isso, esse gradiente também é chamado de **gradiente descendente**.

Considere uma rede simples com 2 pesos e 1 bias. Um resultado possível do vetor gradiente descendente seria, por exemplo, [0,5 ; 50 ; -2,3]. Isso indicaria que à medida que aumentássemos o valor do primeiro e do segundo pesos (já que são positivos) estaríamos caminhando em direção a um erro menor. Da mesma forma,

deveríamos reduzir o valor do terceiro parâmetro (o bias, por exemplo). É interessante notar que o gradiente **não** nos revela o quanto devemos reduzir ou aumentar de um parâmetro, mas sim a sua direção. Contudo, ele pode nos revelar quais os parâmetros são mais impactantes no cálculo final da função de custo. No exemplo anterior, o segundo parâmetro impacta 100 vezes mais a função de custo do que o primeiro parâmetro, visto que ele teve um resultado de 50 versus um resultado de 0,5.

Outro ponto a ser ressaltado é que o cálculo do vetor gradiente descendente deve ser realizado para todas as imagens do nosso conjunto de treinamento, e então deve-se obter um vetor gradiente descendente médio. Contudo, como isso tornaria o processo computacionalmente inviável se tivermos muitas amostras no conjunto de treinamento, a abordagem conhecida como **gradiente descendente estocástico** é utilizada. Ao invés de calcular o vetor gradiente descendente para todas as imagens, o gradiente estocástico separa aleatoriamente o conjunto de treinamentos em subconjuntos, e então calcula o vetor gradiente descendente médio apenas para esse subconjunto (obviamente, os conjuntos vão sendo alternados a cada nova iteração da rede).

O que é o algoritmo de retropropagação então?

O algoritmo de retropropagação é um algoritmo que transforma a função de custo da rede em uma função utilizando pesos, bias e funções de ativação e utiliza a técnica do gradiente descendente estocástico, realizando cálculos de derivadas, para encontrar a direção média (considerando um conjunto de amostras) em que cada parâmetro da rede (pesos e bias) deveria seguir (aumentar ou diminuir) para que a rede caminhe a uma taxa de erro próxima a 0. Depois disso, utiliza o parâmetro de **taxa de aprendizado** (normalmente um valor menor que 1) para transformar o valor médio do vetor do gradiente descendente e atualizar os parâmetros da rede. Lembre-se: a taxa de aprendizado é utilizada pois, caso um valor muito alto fosse usado para atualizar os pesos, a rede poderia nunca chegar a uma convergência.

Outro ponto importante é que para utilizar o gradiente estocástico (que não é tão efetivo quanto o gradiente padrão, mas que garante uma boa aproximação e um grande ganho computacional), as imagens de cada subconjunto devem ser

equilibradas. Por exemplo, o que aconteceria caso o algoritmo de retropropagação apenas utilizasse imagens do dígito 2 para a definição dos pesos? Provavelmente a rede seria especializada em reconhecer dígitos 2, e não funcionaria com os demais.

O algoritmo de retropropagação segue os seguintes passos:

Passo 1: Transforme a função de custo em uma função que utilize pesos, bias e funções de ativação.

Passo 2: Separe o conjunto de treinamento aleatoriamente em subconjuntos, para utilização do gradiente estocástico.

Passo 3: Selecione um subconjunto de treinamento e, para cada amostra, realize a sua predição na rede.

Passo 4: Calcule a função de perda para cada amostra.

Passo 5: Calcule o vetor gradiente descendente para cada amostra;

Passo 6: Calcule o vetor gradiente descendente médio, utilizando o vetor gradiente descendente de cada amostra.

Passo 7: Utilizando a taxa de aprendizado da rede, atualize os pesos e bias.

Passo 8: Repita o passo 3 até que a rede alcance uma boa taxa de acurácia ou um número máximo de repetições.

O funcionamento da rede segue, portanto, basicamente em duas direções, conforme exibida nas Figuras 40 e 41. No primeiro passo, denominado forward, a rede prediz um conjunto de amostras de treinamento e avalia o seu resultado. No segundo passo, denominado retropropagação, a rede atualiza os pesos e bias, baseando-se no vetor gradiente descendente calculado.

Figura 40 – Fase de forward de uma rede feed-forward, onde amostras são estimadas.

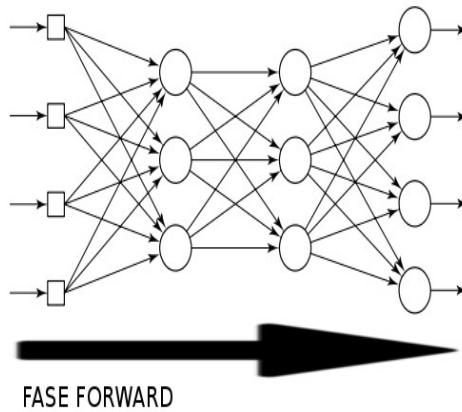
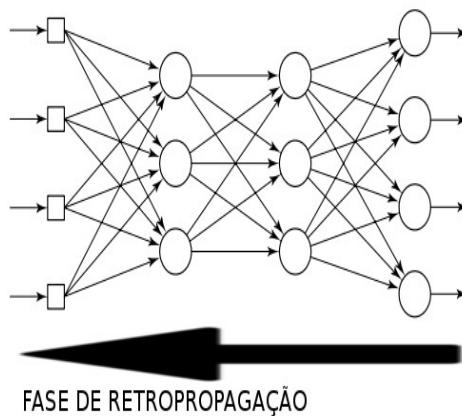


Figura 41 – Fase de retropropagação de uma rede feed-forward, onde pesos e bias são atualizados.



Resumo

Neste capítulo, entendemos o que é o aprendizado de máquina. Vimos que são soluções que utilizam dados históricos para automaticamente conseguirem obter relações entre dados e desenvolver algoritmos. Diferenciamos também aprendizado supervisionado (em que há a presença de dados pré-categorizados), não-supervisionado (onde não há anotações) e semi-supervisionado (em que apenas um

subconjunto das amostras está categorizado). Por fim, entendemos a diferença entre soluções tradicionais de aprendizado de máquina e deep learning, chegando à conclusão de que deep learning obtém características do problema de forma automática, hierárquica, mas que é menos inteligível e que demanda maior capacidade computacional e dados para funcionar bem.

Além disso, abordamos os seguintes tópicos relativos às Redes Neurais Artificiais (RNA):

1) A motivação a partir da qual foram criadas as redes neurais artificiais. As RNAs foram criadas através da constatação de que o cérebro humano era muito eficiente para a realização de determinadas tarefas específicas, como reconhecimento de padrões, atividades difíceis de serem executadas por um computador. Foi proposta então a criação de um modelo computacional que simulasse o funcionamento do cérebro humano, para que fosse possível desempenhar tarefas que ele desempenha bem. Contudo, apesar de ser uma ideia da década de 1940, devido a restrições em recursos computacionais as redes neurais não tiveram uma utilização plena. Seu maior emprego só começou a ser possível por volta de 2009, quando avanços na área de *hardware* tornaram processadores e GPUs mais fáceis de serem obtidos.

2) A estrutura básica de uma RNA é feita para simular a constituição do cérebro humano. Os neurônios são unidades de processamento que recebem informações, as processam e as propagam para outros neurônios. Eles estão divididos em camada, sendo que a primeira camada é conhecida como camada de entrada, a última camada é chamada de camada de saída e as camadas intermediárias são chamadas de camadas ocultas. Os neurônios possuem ligações com neurônios das camadas imediatamente anterior e posterior.

3) Um determinado neurônio recebe informações de vários neurônios da camada anterior. Cada uma dessas informações possui um peso diferente, assim como acontece no cérebro humano. A tarefa do neurônio é agregar essas informações, tipicamente através de um somatório ponderado pelo peso de cada ligação, e somando o resultado a um bias, um valor que tem o mesmo objetivo de

uma constante em uma função matemática. Antes de enviar o resultado obtido para os neurônios da camada posterior, uma função de ativação é aplicada sobre ele. O objetivo é verificar o quanto aquele valor obtido ativou a ligação entre os neurônios e os demais.

4) Uma rede do tipo feedforward é uma RNA que não possui ciclos entre os neurônios, e em que a informação propaga sempre em uma só direção. Os neurônios da camada de entrada são responsáveis por obter as informações de entrada e propagá-los para os neurônios da camada seguinte, que fazem os cálculos e utilizam a função de ativação até que o resultado chegue nos neurônios da camada de saída. Uma RNA em um problema de aprendizado supervisionado aprende com um determinado conjunto de treinamento, uma série de amostras cujo resultado já se conhece. O processo de treinamento da rede envolve processar as amostras do conjunto de treinamento e comparar o resultado obtido com o resultado previamente conhecido, através de uma função de perda. A função de custo é a agregação da função de perda de todas as amostras individuais. Por fim, a taxa de aprendizado determina qual a velocidade com que a RNA irá mudar o conhecimento que já adquiriu, sendo necessário um valor intermediário desse parâmetro, de forma que a rede consiga produzir um resultado útil, porém não leve muito tempo para terminar de ser treinada.

5) O algoritmo de retropropagação é um algoritmo utilizado para atualizar os parâmetros de uma rede MCP. Ele utiliza o conceito de gradiente matemático, que indica em qual direção obtém-se o maior crescimento de uma função. Ao se modelar a função de custo de uma rede como uma função, é possível utilizar o algoritmo de retropropagação para calcular a direção de atualização de cada parâmetro de forma que o erro de treinamento da rede aproxime-se de zero. O gradiente descendente estocástico é um método que otimiza o tempo de treinamento da rede, realizando o cálculo do gradiente para apenas um subconjunto das amostras.

Capítulo 6. Redes Neurais Convolucionais (CNN)

Neste capítulo abordaremos a principal rede utilizada nos processos de visão computacional, as chamadas Redes Neurais Convolucionais (CNN, do inglês *Convolutional Neural Network*). Estudaremos em detalhes o seu funcionamento, os elementos que as constituem e o que cada parâmetro dentro dessas estruturas podem produzir. Abordaremos também o Keras, framework que iremos utilizar para realizar nossos primeiros experimentos utilizando as redes convolucionais.

Introdução às Redes Neurais Convolucionais (CNN)

Como vimos no capítulo anterior, os métodos tradicionais de visão computacional possuem algumas limitações que estão sendo superadas por soluções baseadas em deep learning. O principal tipo de rede responsável por realizar processamentos com visão computacional em deep learning é o conhecido como Rede Neural Convolucional (CNN).

Mas por que não utilizar redes tradicionais, totalmente conectadas, para realizar tarefas de visão computacional? Assim como nas soluções tradicionais não é eficaz utilizarmos os pixels puros da imagem, em soluções com redes neurais também não.

Vamos supor que desejamos desenvolver uma solução de deep learning para a classificação de imagens representando dígitos escritos à mão, conforme mostra a Figura 42.

Figura 42 – Exemplos de dígitos escritos à mão.



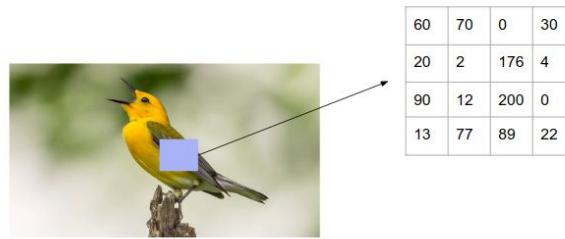
Se utilizarmos uma rede totalmente conectada para resolver esse problema, cada pixel da imagem seria associado a um neurônio na camada de entrada da rede, que estavam por sua vez conectados a todos os neurônios da camada posterior e assim sucessivamente.

Porém, essa abordagem traz um problema: à medida que o número de camadas aumenta, o número de parâmetros também, e a rede demora mais tempo para ser treinada. Além disso, se precisássemos processar imagens com uma resolução espacial muito alta, como a coletada por drones, esse processo poderia ficar inviável, mesmo com a utilização de GPUs.

A solução, proposta em 1998, foi a utilização de uma arquitetura diferente de rede neural, especializadas para soluções de visão computacional. Essas redes são baseadas em uma série de convoluções matemáticas, e, portanto, receberam o nome de Redes Neurais Convolucionais (CNN).

Ao invés de ligar pixels aos neurônios, a CNN trabalha de uma vez com um conjunto de pixels, em uma estrutura conhecida como campo receptivo, conforme mostra a Figura 43. Esse campo receptivo é uma espécie de janela deslizante, que se inicia no começo da imagem e a percorre, cada vez capturando um conjunto de pixels diferente.

Figura 43 – Campo receptivo de uma rede CNN.



Cada campo receptivo coletado passa então por um processo de convolução. A convolução matemática com matrizes consiste em uma operação matemática em que os elementos de uma matriz são multiplicados pelos elementos de uma outra

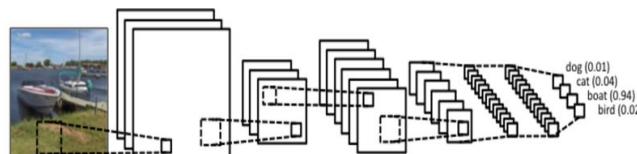
matriz (cada elemento é multiplicado pelo elemento correspondente da outra matriz) e as multiplicações são então somadas, conforme mostra a Figura 44.

Figura 44 – Processo de convolução matemática.

Nas redes CNN, os campos receptivos coletados da imagem são multiplicados por matrizes cujos valores são os parâmetros a serem definidos na rede. O resultado das convoluções dos campos receptivos de uma imagem produz uma outra matriz, menor que a original, conhecida como **feature map** (ou **mapa de características**).

A arquitetura de uma CNN é tradicionalmente representada como na Figura 45. Assim como nas redes tradicionais, elas também são organizadas em camadas, porém cada uma é composta por elementos com funções diferentes e, portanto, recebem nomes distintos. A imagem de entrada é enviada para a primeira camada, que produz uma série de operações sobre ela (por exemplo, as convoluções). O resultado desse processamento (os mapas de características ou feature maps) são enviados para a próxima camada e assim sucessivamente, até que a imagem tenha passado por todas as camadas.

Figura 45 – Representação típica de uma CNN.

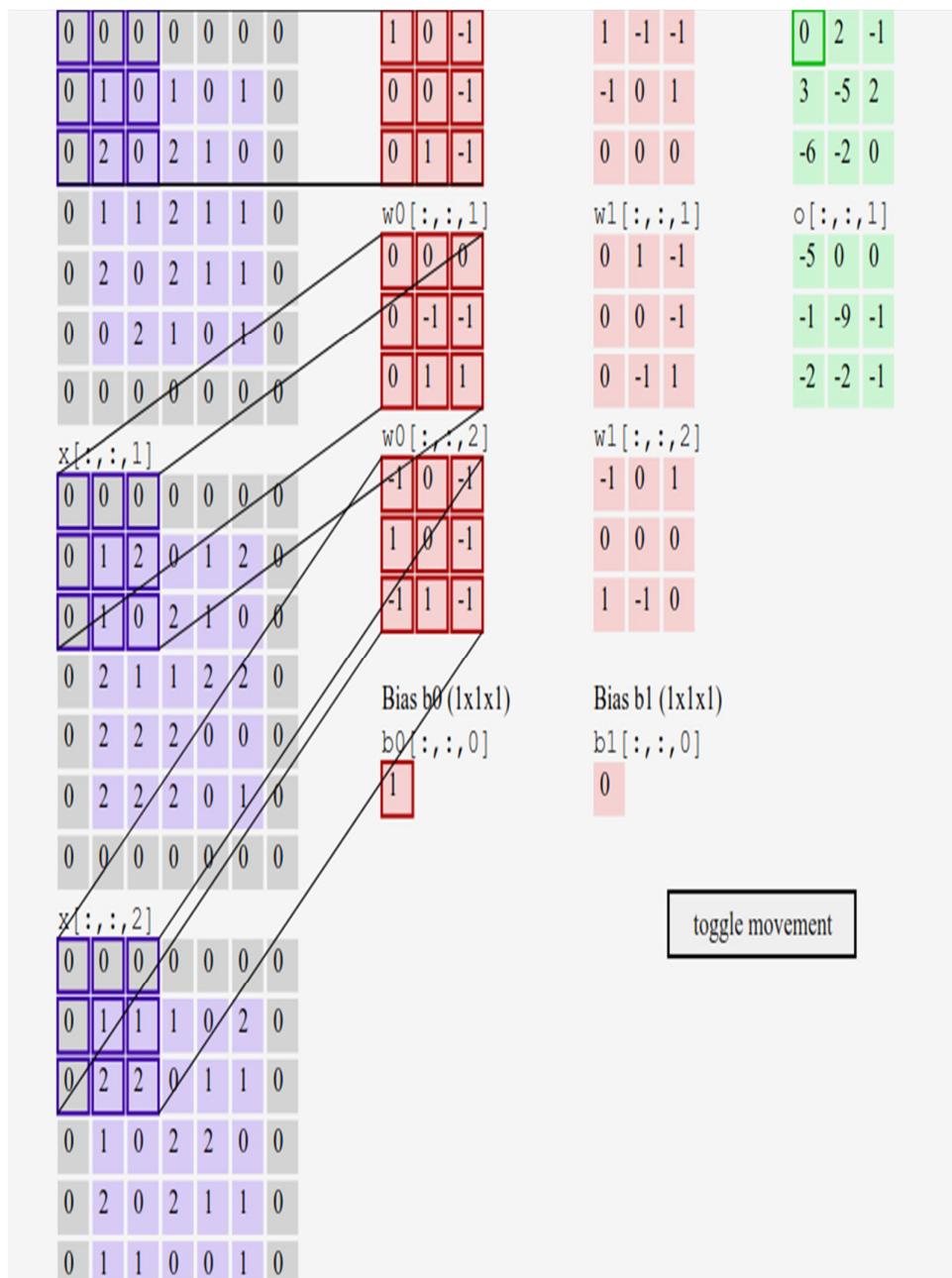


Todas as camadas de uma rede convolucional possuem três dimensões: largura, altura e profundidade. Na primeira camada, essas dimensões correspondem

ao tamanho original da imagem, e a profundidade à quantidade de valores que pixel possui no sistema de cores utilizado. Supondo que a imagem na figura tenha largura e altura de 100 pixels e esteja no sistema RGB (cada pixel possuindo, portanto, 3 valores), as dimensões da primeira camada serão 100x100x3. A última dimensão também é chamada de **canal**. Portanto, nesse exemplo a imagem original possui 3 canais. À medida que a imagem trafega na rede, essas dimensões vão sendo alteradas, conforme veremos a seguir.

O processo de convolução descrito anteriormente também leva em consideração essas três dimensões. Supondo que estamos na primeira camada da CNN, existirão três matrizes de convolução ao invés de apenas uma: uma matriz de convolução para cada elemento da profundidade (uma para o vermelho, outra para o verde e outra para o azul na imagem RGB), conforme mostra a Figura 46 (ferramenta disponível em <http://cs231n.github.io/convolutional-networks/>). Nesse exemplo, existem dois processos de convolução distintos acontecendo, representados pelas matrizes vermelha e rosa.

Figura 46 – Processo de convolução com mais de um elemento na profundidade.



Os resultados das convoluções efetuadas em todos os canais são somados, a eles é acrescentado um **bias**, e o resultado vira um elemento na matriz que corresponderá ao **mapa de características** que será enviado à próxima camada. No

exemplo da figura, os elementos em azul correspondem à imagem original, os elementos em vermelho e rosa às matrizes de convolução e as matrizes em verde ao mapa de características. Cada matriz de convolução também é conhecida como **kernel**, ou **filtro**, e os seus valores são parâmetros que serão definidos durante o treinamento da rede (assim como o bias).

Assim como nas redes totalmente conectadas, algumas referências também dizem que a CNN possui neurônios. Os neurônios nessas redes são o conjunto de matrizes de convolução (kernels ou filtros) em cada canal. No exemplo anterior, haveria dois neurônios nessa camada: um formado pelas kernels em vermelho e outro formado pelos kernels em rosa.

Utilização do Keras

Antes de entrarmos em detalhes no funcionamento de cada componente da CNN, e nos outros tipos de camadas presentes nessas redes, iremos apresentar o framework utilizado durante o curso para os nossos primeiros experimentos utilizando redes convolucionais: o Keras.

Diversos frameworks foram criados para facilitar a construção e manipulação de modelos de redes neurais. Alguns exemplos incluem o TensorFlow, CNTK e Theano. Contudo, alguns desses modelos podem não ser tão amigáveis a uma primeira vista, muitas vezes necessitando de um conhecimento matemático mais aprofundado para a sua utilização.

Considere por exemplo o algoritmo representado na Figura 47, que consiste em uma rede convolucional simples. Note que foram necessárias mais de 60 linhas de código para escrever todo o algoritmo. A documentação do Tensorflow pode ser obtida em <https://www.tensorflow.org/>.

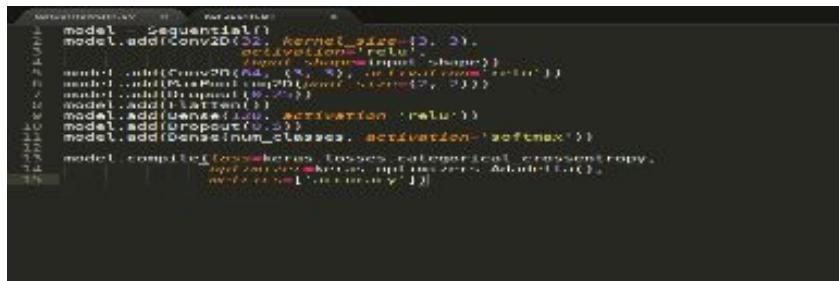
Figura 47 – CNN criada no framework Tensorflow.

```
1 def cnn_model_fn(features, labels, mode):
2     """Model function for CNN."""
3     # Input Layer
4     input_layer = tf.reshape(features["x"], [-1, 28, 28, 1])
5
6     # Convolutional Layer #1
7     conv1 = tf.layers.conv2d(
8         inputs=input_layer,
9         filters=32,
10        kernel_size=[5, 5],
11        padding="same",
12        activation=tf.nn.relu)
13
14    # Pooling Layer #1
15    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
16
17    # Convolutional Layer #2 and Pooling Layer #2
18    conv2 = tf.layers.conv2d(
19        inputs=pool1,
20        filters=64,
21        kernel_size=[5, 5],
22        padding="same",
23        activation=tf.nn.relu)
24    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
25
26    # Dense Layer
27    pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 64])
28
29
30    # Logits Layer
31    logits = tf.layers.dense(inputs=dropout, units=10)
32
33    predictions = {
34        # Generate predictions (for PREDICT and EVAL mode)
35        "classes": tf.argmax(input=logits, axis=1),
36        # Add softmax_tensor to the graph. It is used for PREDICT and by the
37        # `logging_hook`.
38        "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
39    }
40
41
42    if mode == tf.estimator.ModeKeys.PREDICT:
43        return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)
44
45    # Calculate Loss (for both TRAIN and EVAL modes)
46    loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=logits)
47
48    # Configure the Training Op (for TRAIN mode)
49    if mode == tf.estimator.ModeKeys.TRAIN:
50        optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
51        train_op = optimizer.minimize(
52            loss=loss,
53            global_step=tf.train.get_global_step())
54        return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)
55
56    # Add evaluation metrics (for EVAL mode)
57    eval_metric_ops = {
58        "accuracy": tf.metrics.accuracy(
59            labels=labels, predictions=predictions["classes"])}
60
61    return tf.estimator.EstimatorSpec(
62        mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)
```

Por sua vez, o Keras é considerado um framework de **alto-nível**, que abstrai muitas das funções utilizadas em outros frameworks de uma forma mais simples e amigável. Ele pode utilizar o Tensorflow, Theano ou CNTK para executar. Ele permite a criação de modelos de forma mais rápida, oferece suporte a redes convolucionais, pode rodar com a utilização de CPUs ou GPUs e necessita do Python para ser executado. A mesma CNN do exemplo anterior foi reescrita utilizando o Keras na

Figura 48. Note agora que o número de linhas necessárias foi de apenas 15. A documentação do Keras pode ser obtida em <https://keras.io/>.

Figura 48 – CNN criada no framework Keras.



```
1 import tensorflow as tf
2
3 from tensorflow.keras import Sequential
4 from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
5
6 model = Sequential()
7 model.add(Conv2D(32, kernel_size=(3, 3),
8                 activation='relu',
9                 input_shape=(28, 28, 1)))
10 model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
11 model.add(Flatten())
12 model.add(Dense(128, activation='relu'))
13 model.add(Dropout(0.2))
14 model.add(Dense(num_classes, activation='softmax'))
15
16 model.compile(optimizer='adam', loss='categorical_crossentropy',
17                 metrics=['accuracy'])
18
19 print(model.summary())
```

Para a execução dos nossos modelos, podemos utilizar o Google Colab (<https://colab.research.google.com/>). Ele permite a execução de códigos escritos em Python de forma gratuita, atualmente até 12 horas ininterruptas, inclusive com a utilização de GPUs, o que torna os experimentos ainda mais rápidos. Ao entrar no Google Colab, vá até a guia **Edit → Notebook Setting** e selecione a opção **GPU** em Hardware accelerator. Após isso, você já pode executar os seus comandos em Python.

Pré-processamento

Como dito no início dessa apostila, normalmente o processamento de imagens é realizado como uma etapa de pré-processamento de uma visão computacional. Ela consiste em realizar transformações nas imagens de forma que a rede convolucional consiga ser mais eficiente no seu processamento, seja convergindo mais rápido ou chegando a resultados melhores.

Alguns métodos de pré-processamento são tradicionalmente utilizados:

Uniformização de tamanho

Dependendo de como a CNN vai ser criada, ela pode necessitar que todas as imagens estejam do mesmo tamanho para que possam processá-las. Essa etapa

consiste em deixar todas as imagens que serão utilizadas para o treinamento da rede com a mesma quantidade de dimensões, ou seja, o mesmo número de pixels na altura, largura e o mesmo número de canais. Uma estratégia pode ser simplesmente cortar as imagens maiores que o tamanho requerido ou realizar algum processo de amostragem, aumentando ou reduzindo a quantidade de pixels.

Redução de tamanho

Quanto maior é o tamanho (quantidade de pixels) de uma imagem, maior capacidade computacional ela irá requerer para ser processada, tanto em termos de processamento quanto em termos da memória necessária para lidar com ela. A redução de tamanho consiste em reduzir a quantidade de pixels de uma imagem para tornar o processamento mais rápido e leve. Contudo, a redução o tamanho das imagens pode levar a uma perda de detalhes, o que pode comprometer o resultado obtido. Essa, portanto, é uma tarefa que deve ser feita de forma calculada.

Normalização dos valores dos pixels entre 0 e 1

Comentamos que os valores de cada pixel são armazenados entre 0 e 255, pois geralmente são armazenados utilizando o tamanho de um byte. Contudo, uma prática que facilita a convergência da rede é normalizar esses valores entre 0 e 1. Na prática, isso envolve dividir todos os pixels por 255.

Normalização das imagens

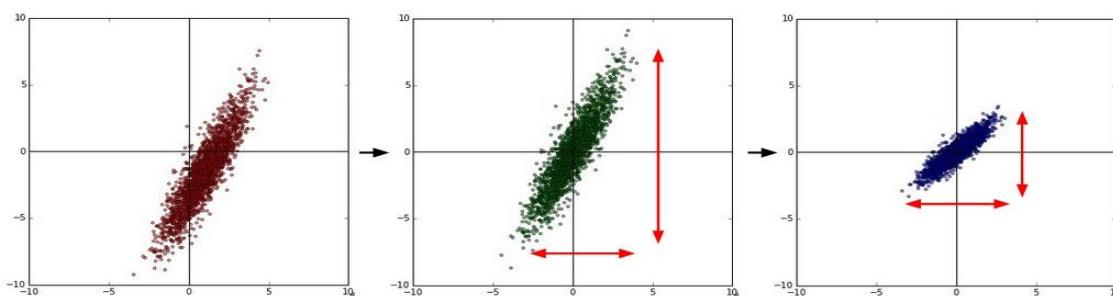
Também com o objetivo de facilitar a convergência da rede, a normalização das imagens pode ser realizada. Isso significa calcular a imagem média (cada pixel será a média do pixel correspondente em cada imagem) e cada imagem de treino passa a ser ela mesma com a subtração da imagem média e divisão pela imagem calcula pelo desvio padrão das imagens, conforme Figura 49.

Figura 49 – Fórmula de normalização das imagens.

$$z = \frac{x - \mu}{\sigma}$$

Na prática, o processo de normalização tem como objetivo centralizar o centroide das amostras em 0 e normalizar a variação em todas as dimensões. Como resultado, a convergência da rede é facilitada. A Figura 50 mostra um conjunto de amostras originais em duas dimensões (vermelho). Cada amostra é reduzida da média de todas elas (verde), o que centraliza a média, e o resultado é dividido pelo desvio-padrão (azul), o que normaliza a variação nas duas dimensões. Para o processo de imagens, o mesmo processo acontece. Contudo, uma imagem possui muito mais dimensões que uma amostra 2D (cada pixel de uma imagem é considerado uma dimensão).

Figura 50 – Processo de normalização em uma distribuição 2D.



Camada convolucional

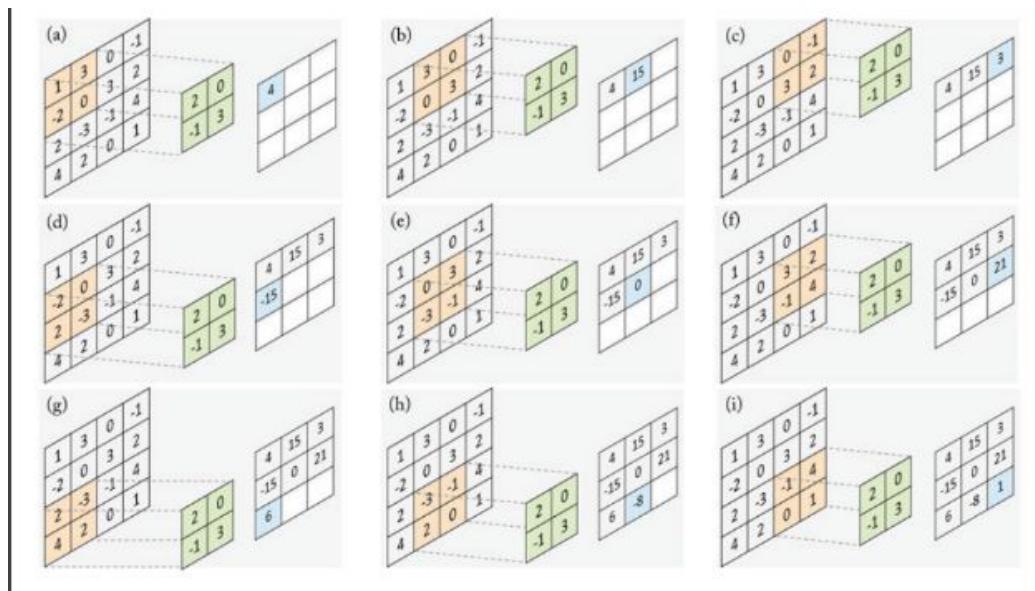
Após as imagens serem pré-processadas, elas estão aptas a passarem pelo processamento da rede convolucional.

Portanto, iremos tratar agora em detalhes de cada componente que compõe uma CNN. Talvez a camada mais importante, e que dá nome a essas redes, seja camada convolucional. Ela é responsável por receber a imagem, aplicar um conjunto de convoluções através dos filtros e produzir como resultado o mapa de características, processo que vimos no início desse capítulo.

Esse processo é exibido novamente na Figura 51. Nela, a imagem original é representada pela matriz maior, em cinza, e o filtro é representado pela matriz com elementos em verde. O campo receptivo em que é feita a convolução é representado

a cada etapa pela matriz laranja, sobreposta à matriz cinza. Perceba que a cada etapa um novo valor é inserido na matriz à direita, com elementos em azul. Essa matriz representa o mapa de características gerado após todas as convoluções terem sido realizadas. Note também que nesse exemplo há apenas um canal (a matriz original tem a profundidade igual a 1), e só é utilizado um filtro (só há uma matriz a partir da qual são feitas as convoluções), e que a soma com o bias também não está sendo representada.

Figura 51 - Processo de convolução realizado em uma camada convolucional.



Porém, como visto na Figura 46, uma camada convolucional pode ter mais de um filtro. Esse é na verdade um **hiper parâmetro** da camada convolucional. Dessa forma, o mapa de características de saída terá um número de canais igual ao número de filtros da camada convolucional (o processo de convolução de cada filtro dará origem a um canal). Outro hiper parâmetro da camada convolucional é o tamanho dos filtros utilizados (no exemplo, o filtro tem o tamanho de 2x2). Filtros menores tendem a capturar mais padrões naquela região da imagem, porém requerem um maior tempo de processamento.

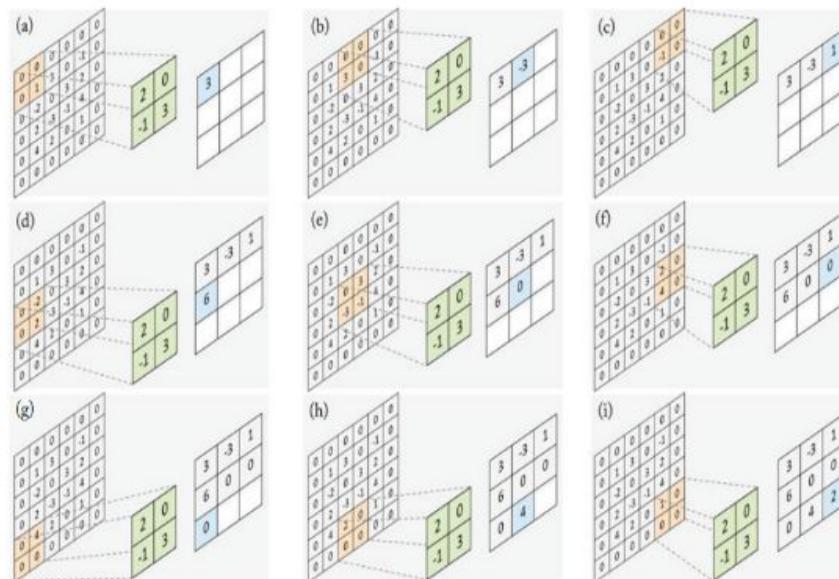
Outro hiper parâmetro da camada convolucional é conhecido como **passo** (ou stride). Ele representa o passo que o filtro dará a cada convolução na imagem de

entrada. Esse valor pode ser configurado para a dimensão horizontal e vertical. No exemplo anterior, esse parâmetro foi definido como 1, ou seja, o filtro pulava de um em um pixel entre uma convolução e outra, tanto na horizontal quanto na vertical. Caso ele fosse definido como 2, o filtro só iria executar quatro convoluções na imagem original, e menos informações seriam mantidas para a próxima camada.

Note que o tamanho do mapa de características da saída é influenciado pelo tamanho do filtro e pelo parâmetro passo. Quanto maiores forem esses dois parâmetros, menores serão as dimensões da saída, visto que menos convoluções serão executadas.

Um outro hiper parâmetro das camadas convolucionais é o **preenchimento**. Ele consiste em aumentar a imagem original com algum valor (normalmente com zeros), de forma que o tamanho da imagem resultante não necessariamente diminua. Esse processo é ilustrado na Figura 52. Perceba que zeros foram adicionados como bordas da imagem, o que consequentemente aumentou o tamanho do mapa de características da saída. Esse parâmetro pode ser importante para manter o tamanho da entrada igual à saída e fazer com que a rede capture informações relevantes nas regiões de borda da imagem.

Figura 52 – Processo de convolução utilizando preenchimento.



A Figura 53 mostra camadas convolucionais criadas utilizando o Keras. Perceba que a primeira camada convolucional possui 32 filtros, cada filtro terá o tamanho de 3x3, o parâmetro passo será 1 na horizontal e vertical e o parâmetro preenchimento (padding) é definido como *same*, ou seja, as dimensões de saída serão iguais às dimensões da entrada. A segunda camada convolucional tem os mesmos parâmetros, exceto pelo preenchimento, que é definido como *valid*, ou seja, não haverá preenchimento.

Figura 53 – Criação de duas camadas convolucionais utilizando o Keras.

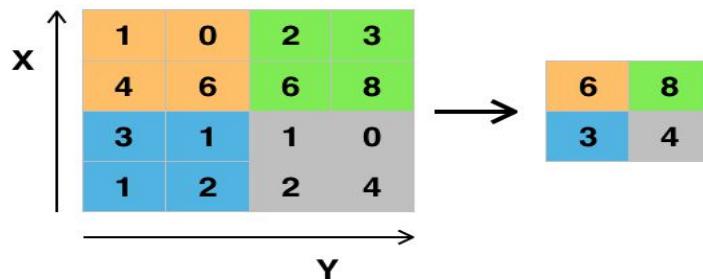


```
cnn.py
1 import keras
2 from keras.models import Sequential
3 from keras.layers import Dense, Dropout, Flatten
4 from keras.layers import Conv2D, MaxPooling2D
5 from keras import backend as K
6
7 model = Sequential()
8 model.add(Conv2D(32, kernel_size=(3, 3), strides=(1,1), input_shape=(32, 32, 3), padding="same"))
9 model.add(Conv2D(32, kernel_size=(3, 3), strides=(1,1), padding="valid"))
10 |
```

Camada de pooling

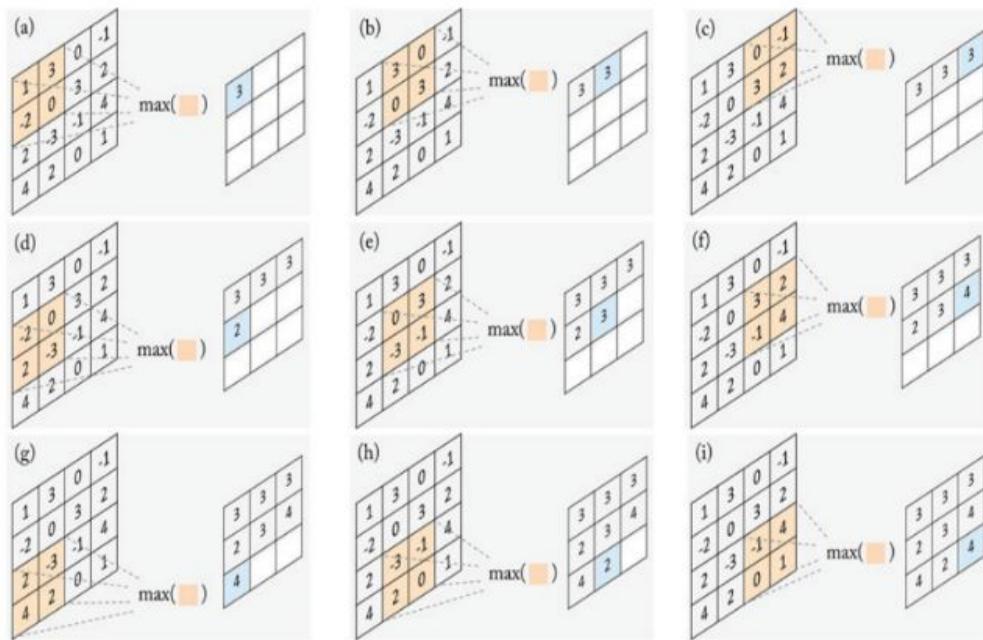
Com o objetivo de tornar as características das imagens invariantes à rotação e a translação, normalmente após cada camada convolucional uma camada especial denominada camada de pooling é adicionada, reduzindo ainda mais o tamanho da imagem. A Figura 54 mostra o processo de uma operação de pooling, utilizando a função max. Nesse exemplo, valores vizinhos são agrupados em cores iguais, e a matriz resultante armazena o maior valor dos vizinhos de mesma cor.

Figura 54 – Exemplo de operação de pooling usando a função max.



As camadas de pooling também possuem o hiper parâmetro que determina o tamanho da matrix de pooling, e o hiper parâmetro passo, que controla quantos pixels a matrix de pooling irá saltar a cada execução. A Figura 55 mostra o processo de operação de pooling em uma imagem de entrada, utilizando uma matrix de pooling de tamanho 2x2 e um passo de 1, além da função max.

Figura 55 – Processo de operação de pooling em uma imagem usando a função max.



Ao invés de utilizar a função max, outra função comumente utilizada é a função média, que consiste, como o nome diz, em se obter a média dos elementos

que estão sendo analisados naquele momento. A escolha de uma ou outra função depende do problema que se esteja analisando, e normalmente é definido de forma empírica. Caso a função max seja utilizada, o resultado tende a extrair as características mais importantes da imagem, como as bordas. Caso a função média seja utilizada, o resultado tende a exibir as características de forma suavizadas. Essa diferença é exibida na Figura 56, em que uma mesma imagem passou pelo processo de pooling utilizando a função max e média.

Figura 56 – Comparação entre a utilização de pooling com a função max (esquerda) e média (direita)



A Figura 57 exibe a criação de duas camadas de pooling utilizando o Keras. Na linha 10, é possível ver a utilização de uma camada de pooling que utiliza a função max, com uma matriz de tamanho 2x2 e um passo horizontal e vertical de 1. Na linha 11, utilizando o mesmo tamanho de matriz e passo, uma camada de pooling que utiliza a função média foi criada.

Figura 57 – Criação de camadas de pooling utilizando o Keras.

```
1 import keras
2 from keras.models import Sequential
3 from keras.layers import Dense, Dropout, Flatten
4 from keras.layers import Conv2D, MaxPooling2D
5 from keras import backend as K
6
7 model = Sequential()
8 model.add(Conv2D(32, kernel_size=(3, 3), strides=(1,1), input_shape=(32, 32, 3), padding="same"))
9 model.add(Conv2D(32, kernel_size=(3, 3), strides=(1,1), padding="valid"))
10 model.add(MaxPooling2D(pool_size=(2, 2), strides=(1,1)))
11 model.add(AveragePooling2D(pool_size=(2, 2), strides=(1,1)))
```

Camadas totalmente conectadas

As camadas exibidas anteriormente (convolucionais e de pooling) são responsáveis por extrair as características das imagens, reduzindo o mapa de características obtidos apenas com as informações relevantes para a sua descrição. Em problemas de classificação, as redes convolucionais geralmente também utilizam camadas totalmente conectadas, assim como as redes neurais tradicionais.

Essas camadas são inseridas geralmente no final das CNNs, e tem como função agregar as características obtidas nas camadas anteriores para tomar uma decisão, por exemplo uma decisão de classificação da imagem. Analisando novamente o desenho típico de uma CNN, na Figura 11, vemos que as últimas três camadas na figura são camadas totalmente conectadas.

Os pixels do mapa de características da camada anterior a elas (até então com três dimensões, altura, largura e profundidade) são transformados em um vetor de uma única dimensão, e elemento desse vetor é associado a um neurônio da primeira camada totalmente conectada. Na prática, cada pixel é associado a um neurônio. Observe, contudo, que o número de neurônios utilizados é muito menor do que se tivéssemos feito esse processo desde o início, com a imagem original, visto que o mapa de características da saída foi reduzido nas camadas convolucionais e de pooling.

Normalmente, a última camada totalmente conectada tem um número de neurônios igual ao número de classes do problema. No exemplo, a última camada totalmente conectada do exemplo possui 4 neurônios, que irão indicar a probabilidade de a imagem pertencer a uma de quatro classes (cachorro, gato, barco e pássaro).

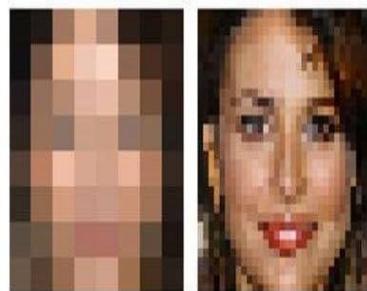
A Figura 58 mostra a criação de uma camada totalmente conectada utilizando o Keras. Perceba que na linha 24 o mapa de características produzido na linha 22 é transformado em um único vetor de 1920800 posições, e então cada uma dessas posições é ligada a um dos 128 neurônios da última camada.

Figura 58 – Criação de uma camada totalmente conectada utilizando o Keras.

```
7 import numpy as np
8 import glob
9
10 fotos = []
11 for index, filename in enumerate(glob.iglob('../lfw/**/*.*')):
12     fotos.append(misc.imread(filename))
13     if (index == 0):
14         break
15
16 fotos = np.array(fotos)
17
18 model = Sequential()
19 # modelo = (250, 250, 3)
20 model.add(Conv2D(32, kernel_size=(3, 3), strides=(1,1), input_shape=(250, 250, 3), padding="valid"))
21 # modelo = (248, 248, 32)
22 model.add(MaxPooling2D(pool_size=(4, 4), strides=(1,1)))
23 # modelo = (245, 245, 32)
24 model.add(Flatten())
25 # modelo = (1920800)
26 model.add(Dense(128))
27 # modelo = (128)
28
29 model.summary()
```

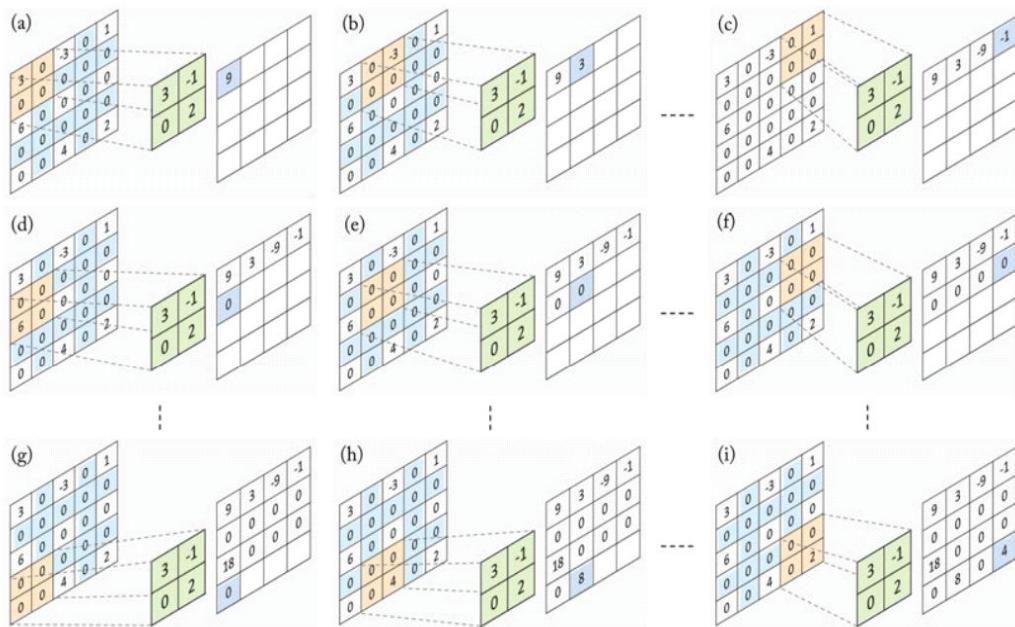
Camadas de convolução transposta

Vimos até aqui que as camadas convolucionais e de pooling geralmente reduzem o tamanho da imagem, produzindo mapas de características com altura e largura geralmente menores (com exceção quando usamos o parâmetro de preenchimento em camadas convolucionais). As camadas de convolução transposta realizam o processo inverso: elas são capazes de aumentar a resolução da saída. Elas são geralmente utilizadas em tarefas em que isso é desejado, por exemplo na reconstrução de imagens com baixa resolução original, conforme mostra a Figura 59.

Figura 59 – Imagem com baixa resolução espacial (esquerda) e restaurada (direita).

Uma camada de convolução transposta (erroneamente conhecida como “deconvolução”), opera da mesma forma que uma camada convolucional, realizando convoluções em uma imagem original. Contudo, ela tem a característica de acrescentar algum valor (geralmente zeros) **entre** os elementos da matriz original. Esse processo é representado na Figura 60. Nessa figura, assim como na camada convolucional, a matriz de verde representa o filtro, e a matriz à esquerda, de cinza, a imagem original. Perceba, contudo, que existem alguns zeros (representados de azul) na matriz original. Esses valores foram inseridos pela camada de convolução transposta para aumentar o tamanho do mapa de características da saída.

Figura 60 – Processo de convolução transposta .



Assim como nas camadas anteriores, um hiper parâmetro da camada de convolução transposta indica o tamanho da matriz de convolução e outro hiper parâmetro indica o tamanho de filtros que serão criados. Contudo, diferentemente das outras camadas, o parâmetro passo na convolução transposta indica quantos zeros serão inseridos verticalmente e horizontalmente. No exemplo anterior, os passos horizontal e vertical foram definidos como 1. Dessa forma, a um zero será inserido entre cada pixel. Caso ele fosse dois, dois zeros seriam inseridos e assim

sucessivamente. Pode-se perceber, portanto, que o parâmetro passo influencia diretamente as dimensões dos mapas de características da saída.

A Figura 61 mostra um exemplo do Keras que cria uma camada de convolução transposta, na linha 25. É possível perceber que essa camada cria 32 filtros, uma matriz de tamanho 3x3 e um passo horizontal e vertical de 2. Da forma como esse exemplo está estruturado, essa camada de convolução transposta irá receber uma entrada de dimensões 245x245x32 e irá produzir como saída um mapa de características de 491x491x32.

Figura 61 – Criação de uma camada de convolução transposta no Keras.

```
1 import keras
2 from keras.models import Sequential
3 from keras.layers import Dense, Dropout, Flatten
4 from keras.layers import Conv2D, MaxPooling2D, Conv2DTranspose
5 from keras import backend as K
6 from scipy import misc
7 import numpy as np
8 import glob
9
10 fotos = []
11 for index, filename in enumerate(glob.iglob('../lfw/**/*.*')):
12     fotos.append(misc.imread(filename))
13     if (index == 0):
14         break
15
16 fotos = np.array(fotos)
17
18 model = Sequential()
19 # modelo = (250, 250, 3)
20 model.add(Conv2D(32, kernel_size=(3, 3), strides=(1,1), input_shape=(250, 250, 3),
21                 padding="valid", activation="relu"))
22 # modelo = (248, 248, 32)
23 model.add(MaxPooling2D(pool_size=(4, 4), strides=(1,1)))
24 # modelo = (245, 245, 32)
25 model.add(Conv2DTranspose(32, kernel_size=(3,3), strides=(2,2)))
26 # modelo = (491, 491, 32)
27
28 model.add(Flatten())
29 # modelo = (1920800)
30 model.add(Dense(128, activation='tanh'))
31 # modelo = (128)
32
33 model.summary()
```

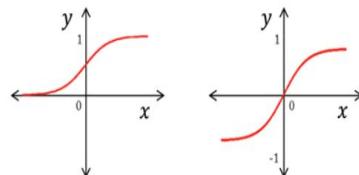
Funções de ativação

Nas camadas convolucionais (incluindo a de convolução transposta), os filtros realizam convoluções em todos os canais, os valores são somados e a eles é acrescentado um bias. Antes, porém, de o resultado ser colocado no mapa de características para ser enviado à próxima camada, uma função de ativação é aplicada sobre ele, assim como ocorre nos neurônios de camadas totalmente conectadas.

Essa função de ativação tem como função acrescentar características não-lineares ao problema, permitindo a modelagem de cenários mais complexos. Caso uma função de ativação não-linear fosse utilizada, toda a rede poderia ser reduzida a uma **única camada convolucional**, independente da profundidade da rede. Portanto, elas representam um componente muito importante no funcionamento das CNNs.

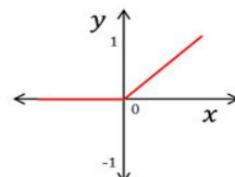
Tradicionalmente, duas funções de ativação eram utilizadas: a função sigmoid e a função tangente hiperbólica (tanh), exibidas na Figura 62. Contudo, à medida que redes mais profundas foram criadas, percebeu-se que essas funções são especialmente suscetíveis ao problema do desaparecimento do gradiente, em que os gradientes das primeiras camadas da rede ficam tão pequenos que os seus parâmetros quase não são alterados.

Figura 62 – Função de ativação sigmoid (esquerda) e tangente hiperbólica (direita).



Para evitar esse problema, uma nova função, menos suscetível a esse problema, começou a se popularizar: trata-se da ReLU, cujo gráfico é exibido na Figura 63.

Figura 63 – Função rectified linear unit (ReLU).



A definição de funções de ativação no Keras também é simples de ser feita, conforme mostra a Figura 64. Note que a função de ativação foi definida tanto para a

camada convolucional quanto para a camada totalmente conectada. Ela também pode ser definida, da mesma maneira, para as camadas de convolução transposta.

Figura 64 – Algoritmo no Keras para configuração de funções de ativação.

```
7 import numpy as np
8 import glob
9
10 fotos = []
11 for index, filename in enumerate(glob.iglob('../lfw/**/*.*')):
12     fotos.append(misc.imread(filename))
13     if (index == 0):
14         break
15
16 fotos = np.array(fotos)
17
18 model = Sequential()
19 # modelo = (250, 250, 3)
20 model.add(Conv2D(32, kernel_size=(3, 3), strides=(1,1), input_shape=(250, 250, 3),
21                 padding="valid", activation="relu"))
22 # modelo = (248, 248, 32)
23 model.add(MaxPooling2D(pool_size=(4, 4), strides=(1,1)))
24 # modelo = (245, 245, 32)
25 model.add(Flatten())
26 # modelo = (1920800)
27 model.add(Dense(128, activation='tanh'))
28 # modelo = (128)
29
30 model.summary()
```

Funções de perda

Assim como redes tradicionais, CNNs também utilizam a função de perda durante o seu treinamento. É a partir dessa função que a rede irá mensurar a qualidade dos resultados obtidos, e irá utilizá-la para guiar o seu processo de treinamento, atualizando os seus parâmetros em direção a um resultado satisfatório.

Para um problema de classificação de imagens, as funções de perda tradicionais de uma rede perceptron de múltiplas camadas podem ser utilizadas também em CNNs. Por exemplo, uma função tradicional é a chamada de erro absoluto médio, exibida na Figura 65. Nessa função, todas as amostras são comparadas com o seu valor esperado, e a diferença entre a predição e o resultado esperado de todas as amostras é somado e dividido pelo total de amostras, indicando o erro da rede.

Figura 65 – Função erro absoluto médio.

$$\frac{1}{n} \sum_{j=1}^n |y_j - \hat{y}_j|$$

Uma variação do erro absoluto médio é o erro quadrático médio, exibido na Figura 66. Ela é similar ao erro absoluto médio, porém a diferença entre as amostras preditas e o valor obtido é elevado ao quadrado. Isso torna essa função menos robusta a ruídos, pois ela penaliza de forma acentuada os erros cometidos durante a classificação.

Figura 66 – Função erro quadrático médio.

$$\frac{1}{N} \sum_{i=1}^N (f_i - y_i)^2$$

Por fim, uma função bastante utilizada em problemas de classificação e que apresenta resultados bastante positivos é a entropia cruzada, exibida na Figura 67. Nessa função, \mathbf{x} representa todas as classes do problema, \mathbf{p} representa o resultado esperado e \mathbf{q} representa o resultado obtido. $P(\mathbf{x})$ representa o valor esperado para a classe \mathbf{x} e $q(\mathbf{x})$ a probabilidade obtida para a amostra pertencer à classe \mathbf{x} .

Figura 67 – Função entropia cruzada.

$$H(p, q) = - \sum_x p(x) \log q(x)$$

Para o cálculo de $q(\mathbf{x})$, ou seja, a probabilidade de que a amostra pertença à classe \mathbf{x} , a função de ativação softmax normalmente é utilizada. Por isso, essa função de perda é também conhecida como “perda softmax”. Como a classificação é executada na última camada totalmente conectada, a função softmax é utilizada nos neurônios dessa última camada (lembre-se que essa camada possui um neurônio para cada classe do problema). A função de ativação softmax é exibida na Figura 68.

Nessa função, z_i representa o valor da saída do neurônio referente à classe i , e j representa cada classe (e, portanto, todos os neurônios), e z_j representa o valor da saída do neurônio referente à classe j . A expressão $\exp(x)$ indica o número e elevado a x . Na prática o que a softmax faz é dividir o valor da ativação do neurônio de uma

determinada classe pela soma das ativações de todas as classes, calculando uma probabilidade. Esses valores são elevados a potência de e para facilitar o processo de treinamento da rede.

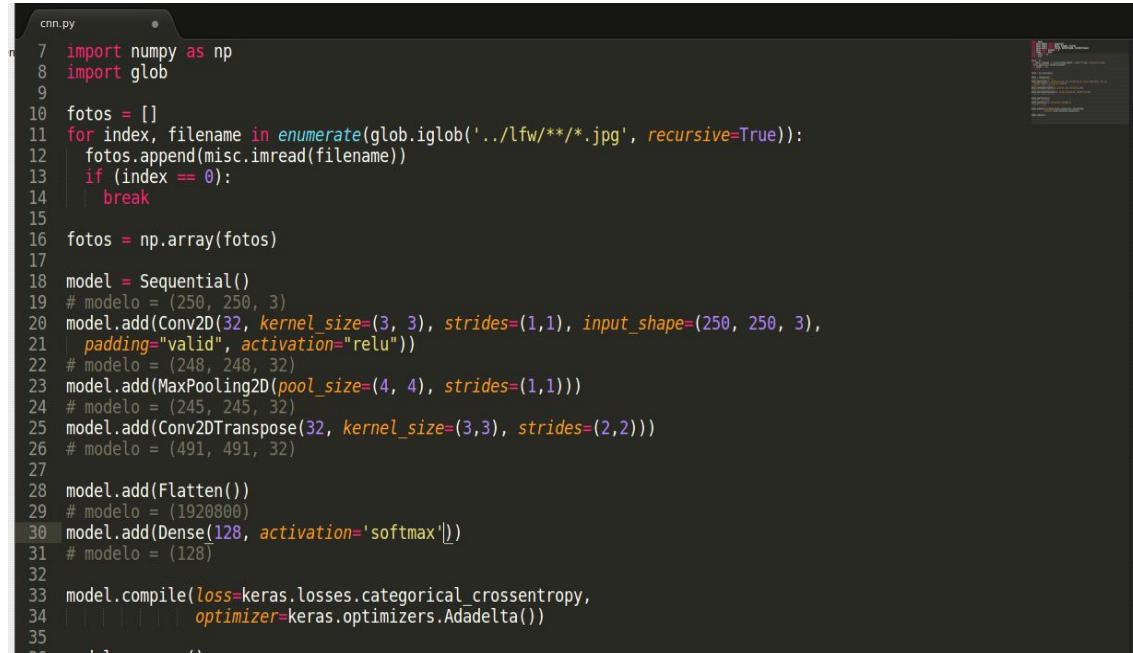
Figura 68 – Função softmax.

$$\frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

É importante ressaltar que tradicionalmente essas funções são utilizadas para problemas de classificação. Caso o problema seja outro, por exemplo a segmentação ou a super-resolução de imagens, as funções de perda poderão ser diferentes.

A Figura 69 exibe a definição da função de perda entropia cruzada no modelo criado no Keras (linha 33). Perceba também que na linha 30 a função softmax foi usada como função de ativação da última camada totalmente conectada.

Figura 69 – Configuração do Keras da função de perda da rede.



```
cnn.py
1 import numpy as np
2 import glob
3
4 fotos = []
5 for index, filename in enumerate(glob.iglob('../lfw/**/*.jpg', recursive=True)):
6     fotos.append(misc.imread(filename))
7     if (index == 0):
8         break
9
10 fotos = np.array(fotos)
11
12 model = Sequential()
13 # modelo = (250, 250, 3)
14 model.add(Conv2D(32, kernel_size=(3, 3), strides=(1,1), input_shape=(250, 250, 3),
15 | padding="valid", activation="relu"))
16 # modelo = (248, 248, 32)
17 model.add(MaxPooling2D(pool_size=(4, 4), strides=(1,1)))
18 # modelo = (245, 245, 32)
19 model.add(Conv2DTranspose(32, kernel_size=(3,3), strides=(2,2)))
20 # modelo = (491, 491, 32)
21
22 model.add(Flatten())
23 # modelo = (1920800)
24 model.add(Dense(128, activation='softmax'))
25 # modelo = (128)
26
27 model.compile(loss=keras.losses.categorical_crossentropy,
28 | optimizer=keras.optimizers.Adadelta())
29
30
31
32
33
34
35
```

Resumo

Nesse capítulo nós abordamos o funcionamento da principal rede para lidar com processos de visão computacional: a CNN. Vimos que a CNN é formada por uma sequência de camadas sequenciais.

A camada convolucional tem a função de realizar convoluções utilizando filtros com os campos receptivo na imagem original, e produzindo como saída mapas de características, que capturam as informações mais importantes da imagem. Nessas camadas, definimos o número de filtros que serão utilizados, o tamanho deles e o parâmetros salto e preenchimento.

As camadas de pooling são responsáveis por reduzir o tamanho da imagem original, tornando o processo invariante a transformações como escala e rotação. Essas camadas possuem como hiper parâmetros o tamanho da matriz utilizada e passos horizontal e vertical. É possível utilizar a função max ou média para reduzir as imagens, cada uma com resultados distintos.

As camadas de convolução transposta também realizam convoluções, porém acrescenta zeros entre os pixels da imagem original, de forma a aumentar o tamanho espacial da saída. Elas também possuem o parâmetro salto, o tamanho da matriz de convolução e a quantidade de filtros a serem utilizados.

As camadas totalmente conectadas geralmente estão presentes no final de uma CNN com atribuição de classificação, e agregam as informações obtidas anteriormente para realizar a classificação. Geralmente, a última camada possui um neurônio para cada classe do problema.

As funções de ativação não-lineares são aplicadas em camadas convolucionais, de convolução transposta e totalmente conectadas com a função de mapear cenários complexos e atribuir características não-lineares ao mapeamento realizado. Caso não fossem utilizadas funções não-lineares, todas as camadas convolucionais poderiam ser reduzidas a apenas uma.

Vimos também que funções de perda conduzem a rede durante o processo de treinamento, mensurando os resultados obtidos. Para o problema de classificação, uma função que apresenta bons resultados é a entropia cruzada, que geralmente é utilizada juntamente com a função softmax na última camada totalmente conectada.

Apresentamos também o Keras, um framework de alto nível para a criação de redes neurais, e o Google Colab, ferramenta que permite a execução de modelos utilizando GPUs de forma gratuita.

Capítulo 7. Aplicações

A visão computacional é uma área de pesquisa bastante ampla e trata de uma série de tarefas diferentes e que apresentam complexidades complementares. Talvez a mais popular dessas aplicações seja o problema de classificação, que consiste em atribuir um, ou mais de um, rótulo a uma determinada imagem, dependendo do seu conteúdo.

Esse capítulo trata de algumas outras aplicações também importantes de serem entendidas, e que possuem desafios específicos. Várias arquiteturas de CNNs estão sendo propostas a cada dia para lidar com essas tarefas, portanto fazer uma revisão da literatura do que se tem disponível é uma tarefa muito complexa. O objeto aqui é prover uma descrição vários problemas, e como soluções de deep learning podem ser utilizadas para resolvê-los.

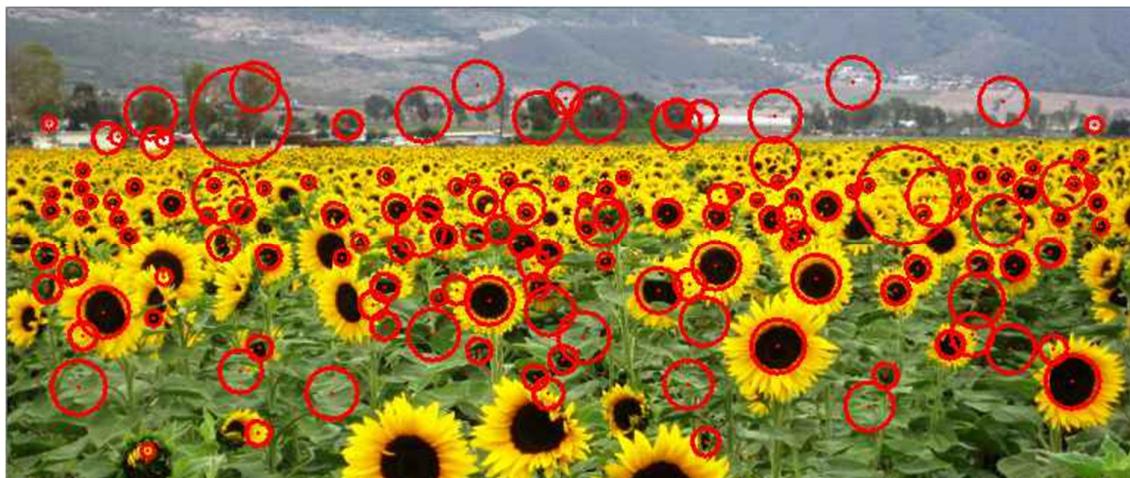
Classificação

Talvez a tarefa mais comum de visão computacional seja a classificação. Ela consiste em atribuir um rótulo (ou vários) a uma imagem, dependendo do seu conteúdo. Em outras palavras, o objetivo é dizer qual objeto a imagem representa. Vimos que, nas redes convolucionais, a tarefa de realizar a classificação é de responsabilidade da última camada totalmente conectada na rede, que normalmente possui um número de neurônios igual ao número de classes do problema.

Apesar de intuitiva, a tarefa de classificação apresenta alguns desafios que devem ser superados. Por exemplo, muitas das vezes as imagens pertencem a uma mesma classe, porém apresentam ângulos de visão diferentes, dependendo da forma que a imagem foi obtida. Muitas das vezes também as imagens apresentam iluminações diferentes, algumas com muita luz ou brilho, enquanto outras estão mais escurecidas. A diferença na escala também pode ser um desafio, visto que um objeto pode estar presente em vários tamanhos em uma imagem, conforme mostra a Figura

70. Deformação dos objetos e parte do objeto em oclusão também são desafios que precisam ser contornados.

Figura 70 – Mesmo objeto, girassol, de vários tamanhos em uma imagem.



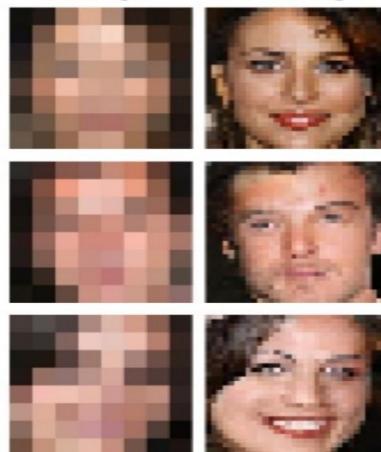
Quando utilizamos deep learning, existem duas formas essenciais de resolvemos esses problemas: a primeira consiste em criar um conjunto de dados bastante completo, com imagens com todas essas características. Pode-se usar a técnica de *data augmentation* para facilitar esse processo. A segunda forma é a realização de um pré-processamento eficiente nas imagens antes da utilização na rede, com o objetivo de remover ruídos e realizar regularização de iluminação, por exemplo.

Um outro desafio dentro da tarefa de classificação é o que se conhece como classificação de alta granularidade. Nesse tipo de problema, classes diferentes apresentam uma baixa variância, ou seja, os objetos de classes diferentes são muito parecidos. Isso acontece, por exemplo, quando queremos categorizar um pássaro na sua espécie específica. Como os pássaros são muito parecidos, todos os objetos de uma mesma classe possuem aspectos em comum.

Super-resolução

Um outro problema que vem ganhando popularidade devido à capacidade computacional das redes convolucionais é o problema da super-resolução. Aqui, o objetivo é aumentar a resolução espacial de uma imagem, ou seja, dada uma imagem com uma resolução baixa quer-se obter a sua imagem correspondente com uma alta resolução, conforme mostra a Figura 71.

Figura 71 – Problema da super-resolução de imagens.



Esta é uma aplicação bastante relevante nas áreas de segurança e forense, por exemplo, onde se tem a foto de uma pessoa com baixa qualidade e se quer prever de fato quem aquela pessoa é. A estratégia básica para lidar com esse problema é a seguinte: obtém-se um conjunto de imagens em alta resolução, e artificialmente reduz-se as suas resoluções. Utiliza-se então como entrada a imagem em baixa resolução, e realiza-se o treinamento da rede esperando que ela retorne a imagem original em alta-resolução, que já se conhece.

Classificação + localização

Na tarefa de classificação, queremos classificar a imagem em uma das classes disponíveis no nosso problema. No problema de classificação + localização, além de classificarmos a imagem queremos saber exatamente onde o objeto se

encontra, geralmente desenhando um retângulo, ou *bounding box*, ao redor dele, conforme representa a Figura 72.

Figura 72 – Resultado da tarefa de classificação + localização.

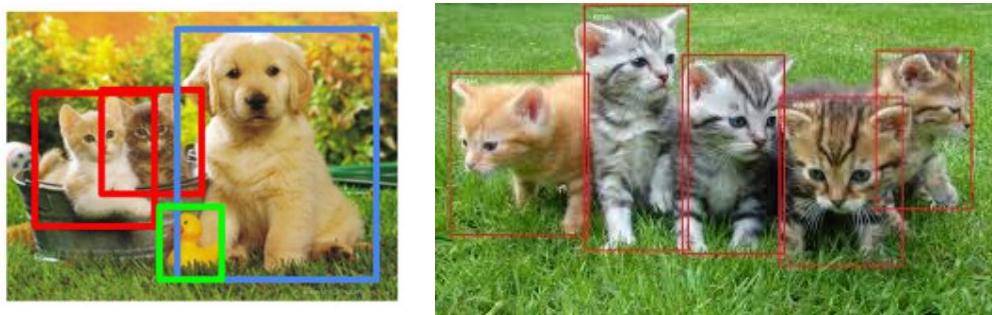


Uma característica importante nesse problema é que se sabe previamente o número de objetos que cada imagem contém, sendo necessário apenas classificá-los e localizá-los. A estratégia principal para solucionar esse tipo de problema é criar uma CNN que irá processar a imagem, mas no final irá possuir dois ramos distintos com camadas totalmente conectadas: uma responsável por classificar a imagem, ou seja, possuindo um neurônio para cada classe, e outro ramo responsável por informar as coordenadas a região do objeto. A função de perda da rede deve levar em consideração ambas as saídas.

Detecção de objetos

Uma tarefa mais complexa que a classificação + localização é detecção de objetos: nesse problema, não se sabe previamente quantos e nem quais objetos existem na cena. Ou seja, a solução deve ser capaz de descobrir se existem objetos, quantos existem, a quais categorias eles pertencem e onde exatamente na imagem cada um está, conforme mostra a Figura 73. Nesse tipo de problema, não é possível criar uma CNN com uma rede totalmente conectada no final, visto que o número de objetos na saída é variável, assim como o número de contornos.

Figura 73 – Exemplos do resultado da detecção de objetos.



Uma primeira estratégia para resolver esse problema é utilizar uma janela deslizante, que irá percorrer toda a imagem e classificar cada bloco, na tentativa de encontrar algum objeto ali. Contudo, essa solução é ineficiente, pois é necessário realizar o processamento a várias regiões diferentes da imagem e utilizar janelas deslizantes de vários tamanhos diferentes, visto que os objetos podem possuir diversos tamanhos.

Uma segunda solução, e que tem apresentando resultados melhores, é a utilização de algoritmos que varrem a imagem na procura de regiões que possam conter objetos. Essa busca é feita considerando aspectos como variações de bordas. Dessa forma, ao invés de se tentar classificar uma quantidade enorme de regiões, o espaço de pesquisa é limitado. Após isso, essas regiões são passadas por redes convolucionais que identificam os objetos. Um exemplo desse método é o R-CNN.

Segmentação semântica

No processo de segmentação semântica, o objetivo é separar cada pixel da imagem em grupos, de acordo com a sua relação semântica. Por exemplo, pode-se desejar separar os pixels de uma imagem em fundo, cavalo, ou jóquei, conforme mostra a Figura 74. Essencialmente, trata-se de um problema de classificação de cada pixel da imagem em uma classe.

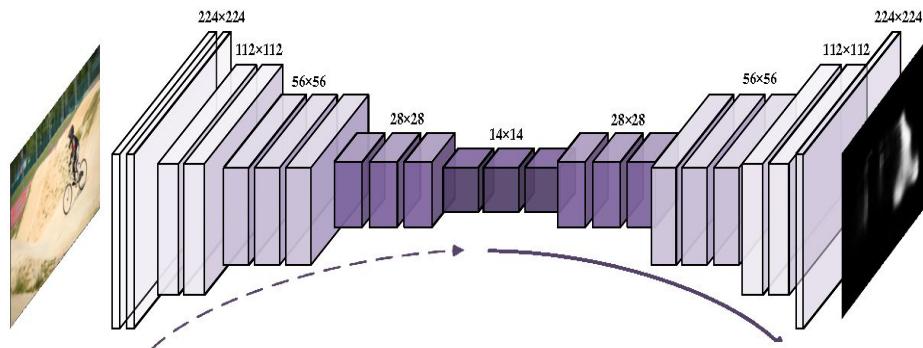
Figura 74 – Problema de segmentação semântica.

A primeira solução é utilizar uma janela deslizante que percorre toda a imagem, denominada janela de contexto. Essa janela é então passada para uma CNN, que a classifica. Considera-se que a classificação obtida é a classificação do pixel central da janela de contexto. O processo é repetido até que todos os pixels tenham sido classificados. O problema dessa solução é que ela é computacionalmente ineficiente, pois deve-se realizar a classificação de inúmeras janelas de contexto.

Uma outra solução é a utilização de uma CNN que recebe a imagem inteira como entrada e produz como resultado uma outra imagem, porém com os valores dos pixels iguais à classe a que cada um pertence. Porém, essa solução também não é eficiente, visto que trabalha com uma resolução espacial muito alta, já que preserva a resolução espacial da imagem de entrada. Esse processo consome muita memória e muita capacidade computacional.

Uma solução mais robusta e que tem apresentado bons resultados são as redes conhecidas como redes totalmente convolucionais (fully convolutional), que tem um exemplo exibido na Figura 75. Essas redes possuem a mesma ideia da anterior, já que recebem como entrada a imagem original e retornam como saída uma imagem onde cada pixel representa o valor da sua classe. Contudo, elas reduzem a resolução espacial da imagem aos poucos, realizam processamentos com a demanda de uma capacidade computacional menor e depois restauram a imagem à sua resolução original.

Figura 75 – Exemplo de rede totalmente convolucional.



Para retornar a imagem à resolução original, essas redes podem usar ou convolução transposta ou operações de *unpooling*, operações contrárias à operação de pooling. A escolha de uma ou outra depende do problema e da capacidade computacional, visto que convoluções transpostas acrescentam mais parâmetros à rede e tornam o treinamento mais difícil.

Um outro desafio no processo da segmentação semântica é o fato de que muitas vezes é difícil obter os valores esperados das imagens, já que se precisa de uma imagem em que todos os pixels estejam categorizados com seus respectivos valores de classe. Esse processo é geralmente demorado e, dependendo do cenário, precisa ser feito por um especialista no problema.

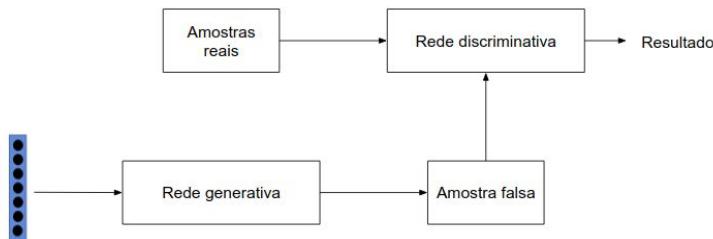
Geração de imagens

Outra tarefa que vem ganhando popularidade é a geração automática de imagens utilizando deep learning. Por exemplo, pode-se querer gerar uma determinada peça de roupa baseada em outras peças de roupas previamente obtidas. Para isso, geralmente utiliza-se uma rede especial, denominada Generative adversarial networks (GAN), propostas em 2014.

Para que possamos adequadamente entender o funcionamento básico de uma rede do tipo GAN, é necessário primeiro diferenciar duas classes de algoritmos: os discriminativos e os generativos. Considerando que um vetor que representa um dado qualquer é denotado por X e que Y denota a classe a que o dado pertence, a primeira classe de algoritmos (discriminativos) procura calcular a probabilidade de o objeto pertencer a uma determinada classe, considerando que se conhece os dados X . Matematicamente, isso é representado por $P(Y/X)$ (ou seja, a probabilidade de uma determinada classe dado o vetor de dados do objeto). Por outro lado, os algoritmos generativos buscam calcular o contrário: a probabilidade de um determinado vetor X dado que se conhece a classe do objeto ($P(X/Y)$).

As redes GAN utilizam desses conceitos para o seu funcionamento. Na verdade, uma rede GAN possui duas redes neurais internamente, uma com caráter generativo e outra com caráter discriminativo. Essa arquitetura é representada na Figura 76. Um vetor de números aleatórios serve como entrada para a rede generativa, que, a partir da utilização de diversos parâmetros, gera uma imagem falsa, com o objetivo de imitar uma distribuição de dados qualquer. Por outro lado, a rede discriminativa recebe essa amostra falsa e um conjunto de amostras reais, e tenta diferenciar uma das outras, com o objetivo de sempre indicar as amostras que são falsas.

Figura 76 – Arquitetura de uma rede GAN.



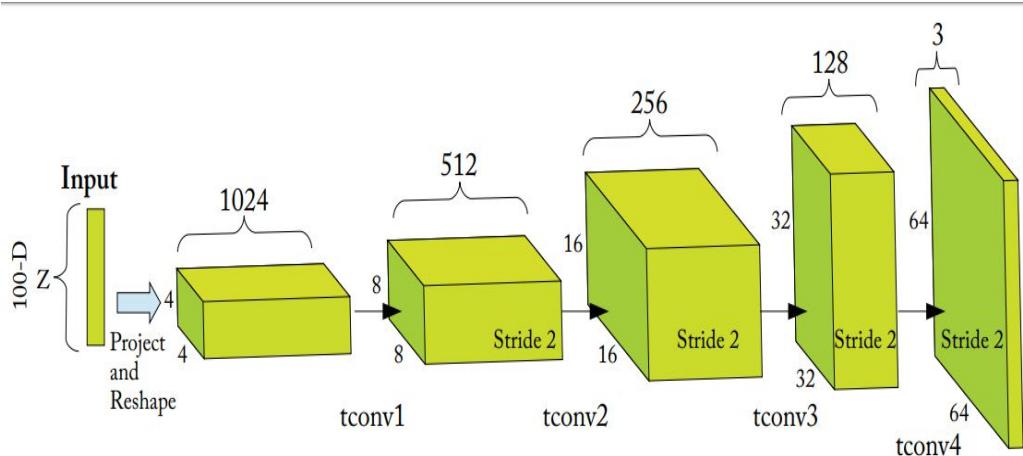
Basicamente, a rede se comporta como uma competição: a rede generativa tenta criar amostras falsas cada vez mais reais, enquanto a rede discriminativa tenta reconhecer essas amostras. A cada época, quando os parâmetros de ambas as redes são atualizados, a competição torna-se cada vez mais desafiadora: a rede generativa

cria amostras cada vez mais parecidas com a distribuição real, enquanto a discriminativa apura cada vez mais sua capacidade de reconhecer dados falsos.

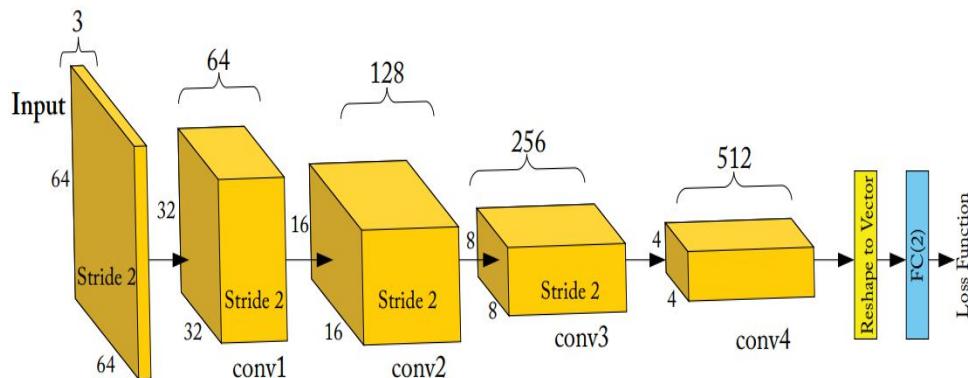
Redes GAN podem ser utilizadas para diversas aplicações, como por exemplo a colorização automática de imagens, geração de imagens a partir simplesmente do seu contorno e na reconstrução de imagens.

A rede Deep Convolutional Generative Adversarial Networks (DCGAN), proposta em 2015, é um exemplo de uma GAN utilizada para a geração de imagens, através da utilização de CNNs. A arquitetura da parte gerativa da rede é exibida na Figura 77. Perceba que a partir de um vetor a rede produz uma imagem, a através da utilização de convoluções transpostas.

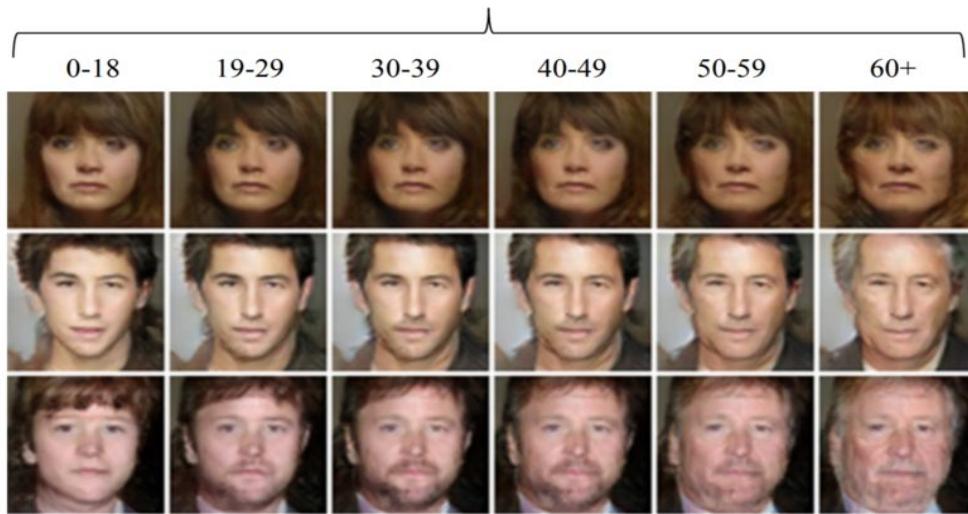
Figura 77 – Rede gerativa da DCGAN.



A rede discriminativa da DCGAN é exibida na Figura 78. Note que a rede recebe a imagem, realiza uma série de convoluções e, no final, possui uma rede totalmente conectada com dois neurônios, que irão indicar se a imagem é falsa ou verdadeira.

Figura 78 – Rede discriminativa da DCGAN.

A Super Resolution Generative Adversarial Network (SRGAN) é uma rede que usa GANs para reconstruir imagens, e a Age-cGan, proposta em 2017, utiliza GANs para criar fotos envelhecidas das pessoas, conforme mostra a Figura 79.

Figura 79 – Resultado da rede Age-cGan.

Resumo

Neste capítulo nós vimos algumas tarefas presentes no universo da visão computacional, e que podem ser solucionados através de redes convolucionais.

Abordamos inicialmente o problema de classificação e os desafios que existem nesse tipo de tarefa, como a classificação de alta granularidade, em que objetos de classes diferentes têm baixa variância. Falamos também do problema de super-resolução, que consiste em aumentar a resolução espacial de uma imagem.

Na tarefa de classificação + localização, o desafio é, ao invés de simplesmente classificar a imagem, informar em qual região ela está, delimitando um contorno para o objeto. Nesse caso, sabe-se previamente o número de objetos na cena. O problema de detecção é ainda mais complexo, pois não se sabe previamente o número de objetos presentes na imagem.

Na segmentação semântica, o desafio é atribuir classes aos pixels, e agrupá-los devido à sua similaridade semântica. Uma solução que vem sendo bastante utilizada são as redes totalmente convolucionais, que reduzem o tamanho da imagem, trabalham com uma resolução espacial pequena e depois restauram o seu tamanho.

As CNNs também estão sendo utilizadas para a geração de imagens, através do conceito conhecido como redes GAN, em que duas redes, uma generativa e outra discriminativa atuam juntas. A DCGAN é composta por duas CNNs que procuram criar imagens de uma determinada distribuição.

Referências

- ABADI, Martín; et al. *Tensorflow: a system for large-scale machine learning*. v. 16. OSDI, 2016.
- ALPAYDIN, Ethem. *Introduction to machine learning*. MIT press, 2014.
- ANTIPOV, Grigory; BACCOUCHE, Moez; DUGELAY, Jean-luc. Face aging with conditional generative adversarial networks. *2017 IEEE International Conference on Image Processing (ICIP)*, Beijing, 2017, p. 2089-2093.
- BAY, Herbert; TUYTELAARS, Tinne; GOOL, Luc Van. Surf: Speeded up robust features. *European conference on computer vision*. Springer, Berlin, Heidelberg, 2006.
- BISHOP, Christopher M.; NASRABADI, Nasser M. Pattern Recognition and Machine Learning. *J. Electronic Imaging*. v. 16, 2007.
- CHOLLET, François. Keras: *The python deep learning library*. Astrophysics Source Code Library, 2018.
- DAVIES, E. R. *Computer Vision. Principles, Algorithms, Applications and Learning*. 5. ed. Elsevier, 2018.
- GONZALEZ, Rafael C.; WOODS, Richard E. *Digital Image Processing*. 2002.
- GOODFELLOW, Ian; et al. *Deep learning*. v. 1. Cambridge: MIT press, 2016.
- GOODFELLOW, Ian; et al. Generative adversarial nets. *Advances in neural information processing systems*, 2014.
- HE, Kaiming; et al. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, 2017, p. 2980-2988.
- JAIN, Anil K. *Fundamentals of digital image processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

KHAN, Salman, et al. A guide to convolutional neural networks for computer vision. in *A Guide to Convolutional Neural Networks for Computer Vision*, Morgan & Claypool, 2018.

LEDIG, Christian; et al. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, p. 105-114.

LONG, Jonathan; SHELHAMER, Evan; DARRELL, Trevor. Fully convolutional networks for semantic segmentation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015, p. 3431-3440.

LOWE, David G. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*. v. 2. Kerkyra, Greece, 1999, p. 1150-1157.

NEGNEVITSKY, M. *Artificial Intelligence: A Guide to Intelligent Systems*. Addison-Wesley, 2002.

RADFORD, Alec; METZ, Luke; CHINTALA, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint*, 2015.

SIMONYAN, Karen; ZISSERMAN, Andrew. Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, 2014.

SOLEM, J. E. Programming Computer Vision With Python. *Creative Commons*, 2012.

UIJLINGS, J.R.R.; SANDE, K.E.A. van de; GEVERS, T.; SMEULDERS, A.W.M. Selective search for object recognition. *International journal of computer vision*, 2013. p. 154-171.

VENKATESAN, Ragav; LI, Baoxin. *Convolutional Neural Networks in Visual Computing: A Concise Guide*. 1. ed. CRC Press, 2017.

ZAKI, Mohammed J.; JR, Wagner Meira; MEIRA, Wagner. *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

ZHANG, Yinda; et al. Deepcontext: Context-encoding neural pathways for 3d holistic scene understanding. *arXiv preprint*, 2016.

ZHU, Qiang, et al. Fast human detection using a cascade of histograms of oriented gradients. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, New York, NY, USA, 2006, p. 1491-1498.