Capítulo 9

Deep Learning: Uma Introdução às Redes Neurais Convolucionais

Marcos Vinícius dos Santos Ferreira, Kelly Maria da Silva Oliveira, Antonio Oseas de Carvalho Filho e Alcilene de Sousa

Abstract

Convolutional networks have played an important role in the history of deep learning. They are a key example of a successful decomprehension application obtained by studying the brain for machine learning applications. Convolutional networks were also some of the first neural networks to re-solve important commercial applications and remain at the forefront of today's deep learning applications. In this chapter, deep learning will be introduced into convolutional neural networks in theory and practice since the understanding of how convolutional neural networks work with images are not trivial tasks.

Resumo

As redes convolucionais têm desempenhado um papel importante na história do aprendizado profundo. Elas são um exemplo chave de uma aplicação bem-sucedida de compreensão obtidos pelo estudo do cérebro para aplicações de aprendizado de máquinas. As redes convolucionais também foram algumas das primeiras redes neurais a resolver importantes aplicações comerciais e permanecem na vanguarda das aplicações de aprendizado profundo hoje. Nesse capítulo será introduzido o aprendizado profundo abordando as redes neurais convolucionais em teoria e prática visto que o entendimento de como as Redes Neurais Convolucionais trabalham com imagens não são tarefas triviais.

9.1. Introdução

O Aprendizado de Máquina (AM), são explorações que estudam o desenvolvimento de métodos capazes de extrair conhecimento a partir de amostras de dados. Existem diversos algoritmos de aprendizado de máquina cuja finalidade é admitir que, após um determinado

treinamento com certo conjunto de dados cujas instâncias têm classificação conhecida, uma máquina seja capaz de interpretar novos dados e classificá-los de maneira apropriada a partir de uma generalização do que lhe foi oferecido [Libralão et al. 2003]. Existem três paradigmas referentes à forma de aprendizagem:

- **Aprendizado Supervisionado:** utiliza supervisores para obter o melhor modelo desejado na fase de treinamento, ou seja, agente externo que fornece à rede o comportamento desejado com intuito de obter a saída desejada.
- **Aprendizado não supervisionado:** permite que o algoritmo aprenda a reunir entradas segundo alguma medida de similaridade entre as entradas, com intuito de agrupá-las.
- **Aprendizado por reforço:** onde o aprendizado ocorre por meio de recompensas, dependendo do desempenho apresentado.

As Redes Neurais Artificiais (RNAs) inspiram a criação de alguns algoritmos de AM [Libralão et al. 2003]. Uma RNA é uma máquina projetada para modelar a maneira que o cérebro realiza uma tarefa em particular ou função de interesse; a rede é normalmente implementada usando componentes eletrônicos ou é simulada por programação em um computador digital. Conhecidos como nós, as redes neurais se utilizam de estruturas que funcionam semelhantes a neurônios [Haykin 2001].

Uma rede neural é composta por um conjunto extremamente complexo de neurônios e, entre eles, a comunicação é realizada através de impulsos. As redes neurais biológicas não transmitem sinais negativos, ao contrário das redes neurais artificiais, sua ativação é medida pela frequência com que os pulsos são emitidos [Haykin 2001]. As redes naturais não são uniformes como as redes artificiais, e apresentam certa uniformidade apenas em alguns pontos do organismo. Os principais constituintes de um neurônio são ilustrados na Figura 9.1:

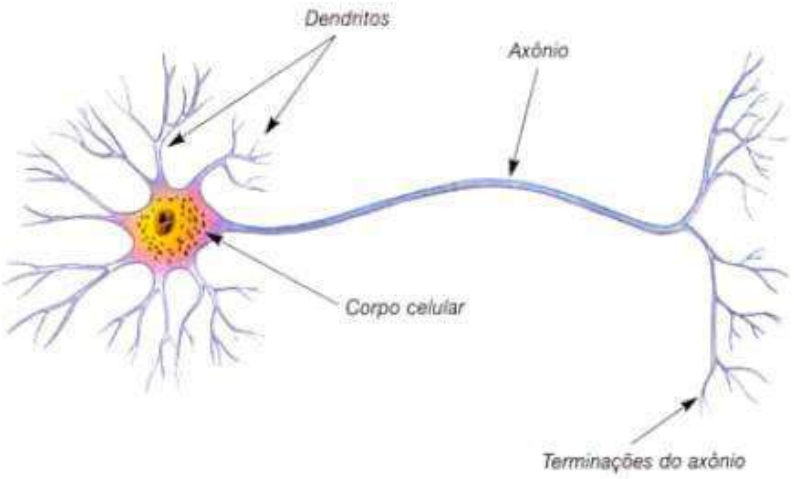

Figura 9.1. Neurônio Biológico. Fonte: [Morais 2017]

- **Dendritos:** responsáveis por receber os estímulos transmitidos pelos outros neurônios;
- **Corpo Celular:** responsável por coletar e combinar informações vindas de outros neurônios;
- **Axônio:** responsável por transmitir os estímulos para outras células (constituído de uma fibra tubular que pode alcançar até alguns metros).

Um neurônio é uma unidade de processamento de informação que é fundamental para a operação de uma rede neural [Haykin 2001]. A Figura 9.2 ilustra os elementos da arquitetura mais comum de uma RNA.

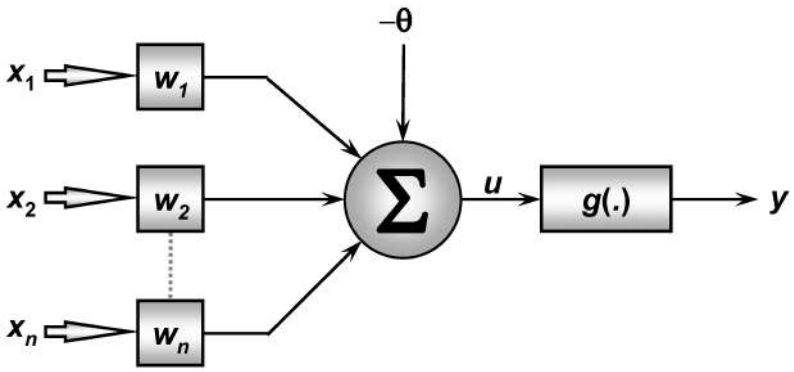

Figura 9.2. Neurônio Artificial. Fonte: [Morais 2017]

- **Entrada:** a camada que produz a saída (resultado) decorrente do meio externo.
- **Sinapses:** ou elos de conexão, emulado através de pesos w1, w2,..., wn acoplados as entradas dos neurônios responsáveis por fazer um valor excitatório ou inibitório.
- **Bias:** ou viés, cujo propósito é permitir uma melhor adaptação da função de ativação, ao qual possui dependência quanto a seu valor ser positivo ou negativo, e que poderá aumentá-lo ou diminuí-lo.
- **Somador:** soma as entradas, ponderadas pelas respectivas sinapses de neurônio;
- **Função de ativação:** se refere a restrição do intervalo permissível de amplitude do sinal de saída a valor finito.
- **Saída:** valor final produzido por um neurônio.

9.2. Convolutional Neural Network

A aprendizagem profunda ou *Deep Learning* é parte de uma família mais abrangente de métodos de AM baseados na aprendizagem de representações de dados. Uma observação, por exemplo, uma imagem, pode ser representada de várias maneiras, tais como um vetor de valores de intensidade por *pixel*, ou de uma forma mais abstrata como um conjunto de

arestas, regiões com um formato particular, etc. Algumas representações são melhores do que outras para simplificar a tarefa de aprendizagem. Uma das promessas da aprendizagem profunda é a substituição de características feitas manualmente por algoritmos eficientes para a aprendizagem de características supervisionada ou semi-supervisionada e extração hierárquica de características [Song e Lee 2013]. A Figura 9.3 faz um breve resumo sobre o surgimento.

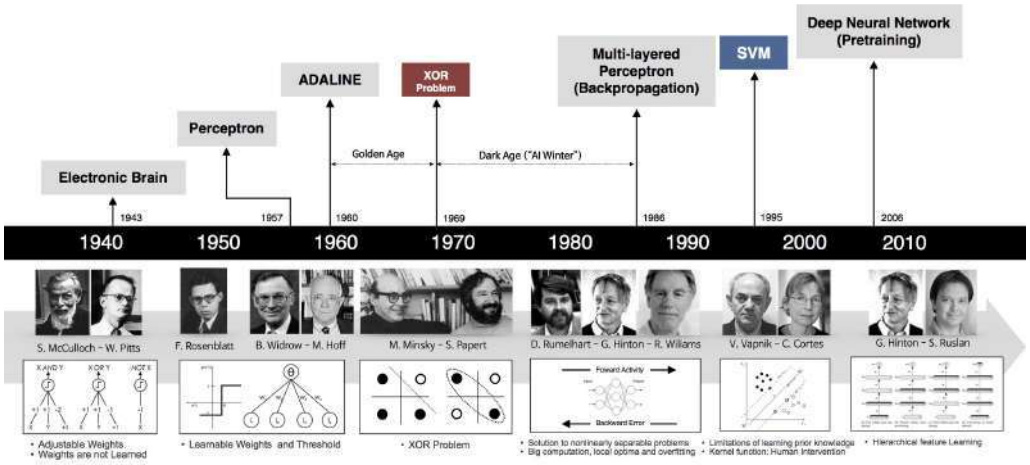

Figura 9.3. Linha do Tempo. Fonte [Beam 2017]

1943 - McCulloch e Pitts projetaram a estrutura que é conhecida como a primeira rede neural, apresentaram um modelo de neurônio como uma unidade de processamento binária e provaram que esta unidade era capaz de executar muitas operações lógicas.

1957 - Rosenblatt introduziu uma nova abordagem para o problema de reconhecimento de padrões com o desenvolvimento do Perceptron [Rosenblatt 1958]. Esta abordagem era a mais simples configuração de RNA. Com apenas uma camada contendo apenas um neurônio. Seu propósito inicial era implementar um modelo computacional inspirado na retina, objetivando-se então um elemento de percepção eletrônica de sinais.

1960- Bernard Widrow e M. Holf, desenvolveram um novo tipo de elemento de processamento de redes neurais chamado de *Adaptive Linear Element* (Adaline). A diferença entre o Adaline e o Perceptron está no procedimento de aprendizagem [Haykin 2001].

1969 - Minsky e Papert analisaram matematicamente o Perceptron e demonstraram que redes de uma camada não são capazes de solucionar problemas que não sejam linearmente separáveis. Tal restrição implicava que padrões de entrada similares que resultase em padrões de saídas similares, levaria o sistema à incapacidade de aprender importantes mapeamentos. A função ou-exclusivo (XOR) é um exemplo clássico deste caso.

1969 a 1982 - um período em que pouco se publicou.

1986 - Rumelhart, Hinton e Williams desenvolveram o algoritmo de treinamento *backpropagation* que permitiu o treinamento eficientemente de redes com camadas intermediárias [Rumelhart et al. 1985]. Este algoritmo resultou nas RNAs mais utilizadas atualmente, as redes *Multi-Layer Perceptron* (MLP), treinadas com o algoritmo *backpro-*

pagation. Nesta arquitetura existem múltiplas camadas de neurônios, sendo que cada um deles está conectado a todos os neurônios da camada seguinte.

1995 – Ascensão da *Support Vector Machines* (SVM).

2006 – Hinton, Simon Osindero e Yee-Whye Teh publicaram um artigo em 2006 que foi visto como um avanço significativo o suficiente para reavivar o interesse em redes neurais. O artigo "Um Algoritmo de Aprendizado Rápido para Redes de Crença Profunda"[Hinton et al. 2006], foi responsável por lgo que mais tarde se transformaria o termo *"Deep Learning"*.

O Aprendizado Profundo permite modelos computacionais que compõem múltiplas camadas de processamento aprender representações de dados com múltiplos níveis de abstração. Esses métodos melhoraram drasticamente o estado da arte no reconhecimento de fala, reconhecimento de objetos visuais, detecção de objetos e muitos outros domínios, como descoberta de medicamentos e genômica. Um dos tipos mais amplamente utilizados de *Deep Learning* são as redes neurais convolucionais ou *Convolutional Neural Network* (CNN) proposta por [LeCun et al. 1990].

As CNNs são uma arquitetura treinável, biologicamente inspirada que possui a capacidade de aprendizado no que se refere a características invariantes [LeCun et al. 2010]. As CNNs são variantes de MLPs inspiradas biologicamente e que consistem em um conjunto de camadas que extraem características de imagens de entrada, fazendo sucessivas convoluções e redimensionando de modo que no final consiga ter apenas a marca da classe a que a imagem de classe pertence.

A arquitetura de uma CNN é um paradigma extremamente versátil, porém conceitualmente simples, que pode ser aplicada a uma extensa gama de tarefas perceptivas. Embora a CNN tradicional treinada com a aprendizagem supervisionada seja muito eficaz, o treinamento pode exigir uma grande quantidade de amostras de treinamento rotuladas [LeCun et al. 2010].

Uma CNN é composta por uma ou mais camadas convolucionais, muitas vezes seguidas de camadas de *Pooling*, e depois seguidas por uma ou mais camadas totalmente conectadas como em uma rede neural multicamada padrão. A arquitetura de uma CNN foi projetada para aproveitar a estrutura 2D de uma imagem de entrada, ou outra entrada 2D, como um sinal de fala. Isto é conseguido com conexões locais e pesos ligados, seguidos de alguma forma de agrupamento que resulta em características invariantes. Outro benefício da CNN é que são mais fáceis de treinar e têm muitos parâmetros menos do que redes totalmente conectadas com o mesmo número de unidades escondidas.

A CNN é uma rede neural que implementa várias camadas distintas, sendo que as principais são as: convolucionais, de *Pooling* e totalmente conectadas ou *Fully-connected*. A camada convolucional tem por funcionalidade extrair atributos dos volumes de entradas. A camada de *Pooling* é responsável por reduzir a dimensionalidade do volume resultante após as camadas convolucionais, ajudando a tornar a representação invariante a pequenas translações na entrada. Por fim, a camada totalmente conectado é responsável pela propagação do sinal por meio da multiplicação ponto a ponto e uso de uma função de ativação. A Figura 9.4 ilustra as três principais camadas da CNN descrita.

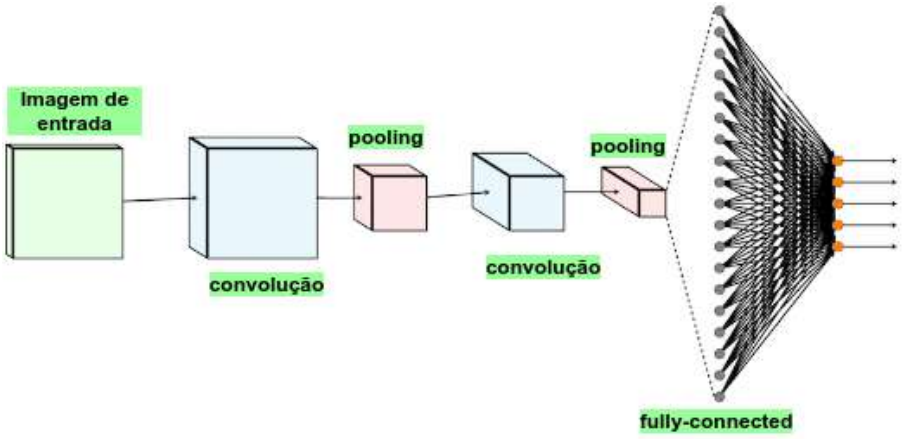

Figura 9.4. Arquitetura de uma CNN. Fonte: Modificada de [Spark 2017]

As CNNs se mostraram eficientes para a Processamento de Linguagem Natural com excelentes resultados na análise semântica, recuperação de consulta de busca, modelagem de sentenças, e outras tarefas tradicionais de Processamento da Linguagem Natural [Kim 2014]. Desde então, uma série de empresas vêm vindo utilizando o aprendizado profundo no centro de seus serviços. O *Facebook* usa redes neurais para seus algoritmos de marcação automática, o *Google* para sua pesquisa de fotos, *Amazon* para suas recomendações de produtos, Pinterest para sua personalização de feed doméstico e Instagram para sua infraestrutura de pesquisa.

9.2.1. Camada Convolucional

A camada Convolucional é o bloco de construção central de uma rede neural convolucional que faz a maior parte do levantamento pesado computacional. Nessa camada é empregada a operação de convolução, que é o tratamento de uma matriz (I) por outra (K) chamada *kernel* que tem por função aplicar filtros na matriz (I) para destacar/mapear características. Nesse tratamento é feito a somatória do produto ponto a ponto entre os valores da matriz *kernel* (K) e cada posição da matriz (I). A Figura 9.5 ilustra o exemplo descrito de uma convolução gerando outra matriz (I*K) com as características mais relevantes.

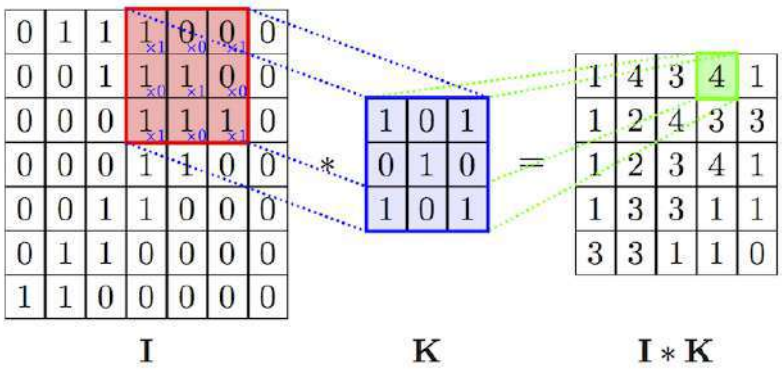

Figura 9.5. Convolução entre as matrizes. Fonte: Modificada de [Spark 2017]

Existem três parâmetros que controlam o tamanho do volume resultante da camada convolucional: profundidade (*depth*), passo (*stride*) e *zero-padding*. Cada um dos filtros será responsável por extrair características diferentes no volume de entrada. Portanto, quanto maior o número de filtros maior o número de características extraídas, porém a complexidade computacional, relativa ao tempo e ao uso de memória, também será maior [Araújo et al.].

9.2.2. Camada de Pooling

Após a camada de Convolução que faz um mapa de características, vem a camada de *pooling* que modifica ainda mais a saída da camada. Essa camada é responsável por reduzir a dimensionalidade de cada mapa de características, mas mantém as informações mais importantes.

A operação de *pooling* pode ser de diferentes tipos: máximo, média, soma etc. No caso máximo, tem-se o *Max Pooling*, definimos um bairro espacial (por exemplo, uma matriz 2×2) e tire o maior elemento do mapa de recursos retificados dentro dessa janela. Em vez de tomar o maior elemento, também poderíamos tomar a média (agrupamento médio) ou a soma de todos os elementos nesta janela. Na prática, o *Max-Pooling* mostrou funcionar melhor. Além de reduzir o tamanho da imagem, consequentemente reduzindo o processamento para as próximas camadas, essa técnica também auxilia no tratamento de invariâncias locais [Ferreira 2017]. A Figura 9.6 demostra como ocorre a operação de -pooling em uma mapa de características.

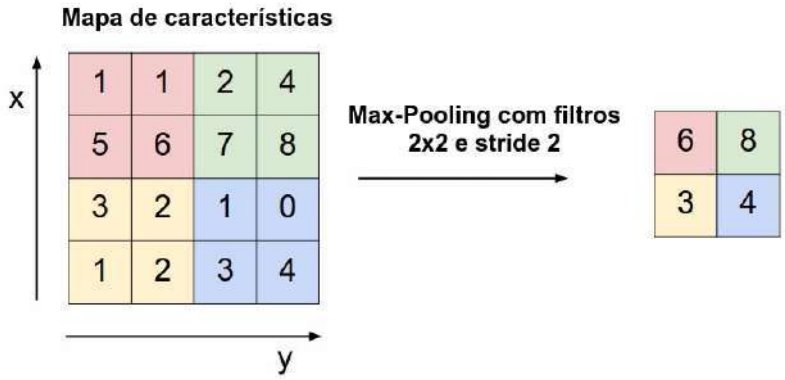

Figura 9.6. Operação de Max-Pooling. Nós deslizamos a matriz de 2×2 por 2 células (também chamado de "stride") e leva o valor máximo em cada região. Modificada de [Deshpande 2016b]

9.2.3. Camada Fully-connected

A camada Totalmente Conectada ou *Fully-connected* é um rede neural Multi Layer Perceptron tradicional que usa geralmente uma função de ativação *softmax* na camada de saída. O termo “Totalmente Conectado” implica que cada neurônio na camada anterior está conectado a cada neurônio na próxima camada.

O resultado das camadas *convolutivas* e de *pooling* é a extração de características de alto nível da imagem de entrada. O objetivo da camada Totalmente Conectada é usar esses recursos para classificar a imagem de entrada em várias classes com base no

conjunto de dados de treinamento.

9.2.4. Arquiteturas

Nos últimos anos, as áreas de aplicação de redes neurais profundas foram rapidamente expandidas. Várias trabalhos envolvendo arquiteturas com uma única abordagem para diferentes problemas, entre as quais podemos destacar: AlexNet (2012), GoogleNet (2014), VGGNet (2014), ResNet (2015).

9.2.4.1. AlexNet (2012)

Em 2012, Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton projetaram a AlexNet [Krizhevsky et al. 2012]. Treinaram uma rede para classificar 1,2 milhões de imagens de alta resolução no concurso *ImageNet Large Scale Visual Recognition Competition* (ILSVRC) 2010 em 1.000 possíveis categorias. A rede era composta por 5 camadas convolucionais, camadas de *max-pooling* e três camadas totalmente conectada com *dropout*. O concurso consiste em vários times que competem para saber quem implementa o modelo mais eficiente para classificação, detecção e localização de objetos em imagens.

O ILSVRC 2012 ficou marcado como o primeiro ano no qual uma CNN atingiu o primeiro lugar desse desafio, com uma taxa de erro de teste de top-5 vencedora de 15,3%, em comparação com 26,2% alcançada pela segunda melhor entrada. A rede AlexNet introduzida em Krizhevsky et al. (2012) foi um marco para a divulgação da CNN na comunidade de visão computacional. Foi a primeira vez que um modelo foi executado tão bem em um conjunto de dados ImageNet historicamente difícil, a mesma foi responsável por influenciar dezenas de outras redes convolucionais para reconhecimento de padrões em imagens.

9.2.4.2. ZF Net (2013)

Após o sucesso da AlexNet em 2012, vários modelos baseados em CNN foram submetidos para o ILSVRC 2013. O vencedor em 2013 foi a rede construída por Zeiler e Fergus, 2013 com um modelo que obteve uma taxa de erro de 11,2%. Esta arquitetura foi concebida a partir de modificações na arquitetura AlexNet, e desenvolveu algumas idéias básicas sobre como o seu desempenho poderia ser melhorado. A maior contribuição de Zeiler e Fergus foi o detalhamento do modelo proposto e o desenvolvimento da técnica DeConvNet (Deconvolutional Network), que consiste na forma de visualização dos mapas de características da rede [Zeiler e Fergus 2014]. A Figura 9.7 ilustra exemplos de imagens de entrada e das regiões que ativam os filtros.

A ideia básica de como isso funciona é que, em todas as camadas da CNN treinada, é anexado uma “DeConvNet” que possui um caminho de volta aos pixels da imagem. A técnica permite operações reversas de pooling até o momento em que a imagem atinja seu tamanho original. Esta operação possibilita a verificação de como cada camada presente na CNN, ler a imagem de entrada e quais regiões ativam os filtros.

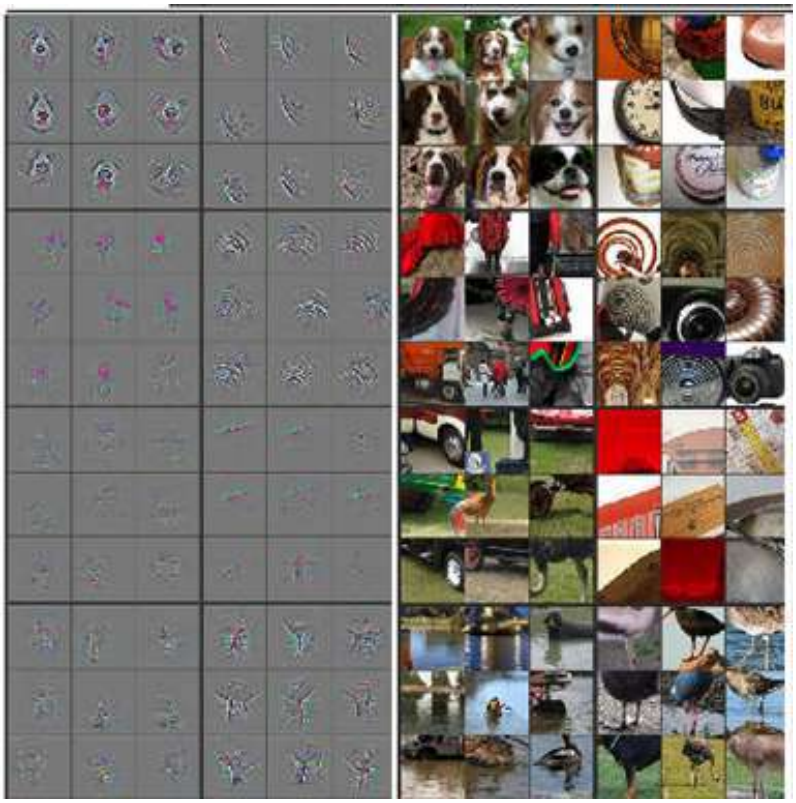

Figura 9.7. Exemplos de imagens de entrada e das regiões que ativam os filtros. Modificada [Deshpande 2016a]

9.2.4.3. GoogLeNet (2015)

GoogLeNet proposto por [Szegedy et al. 2015], foi o primeiro modelo que introduziu a ideia de que as camadas das CNNs poderiam ser executadas paralelamente. A arquitetura do módulo Inception, uma rede de 22 camadas profundas, iniciou um patamar elevado de classificação e detecção no que diz respeito a recursos de reconhecimento visual 3D WideNet 2014 (ILSVRC14) da ImageNet, com uma taxa de erro de 5 maiores de 6,7%.

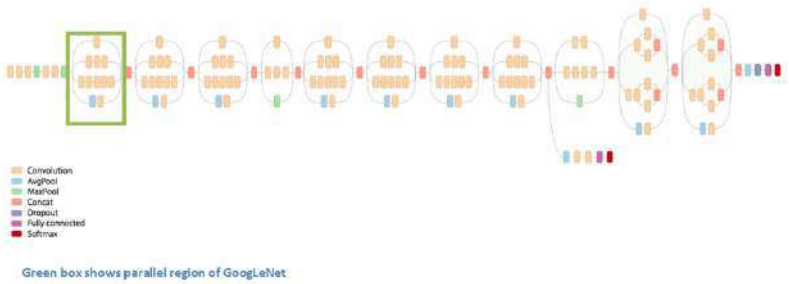

Figura 9.8. Arquitetura da GoogLeNet. Modificada [Deshpande 2016a]

Analisando a Figura 9.8, o módulo de inicialização permite que as execuções de todas as operações sejam executadas em paralelo. Os módulos Inception ilustrados na Figura 9.9 consistem de combinações paralelas de camadas com filtros convolucionais

de tamanho 1×1, 3×3 e 5×5. As camadas 1×1 fornecem um método de redução de dimensionalidade.

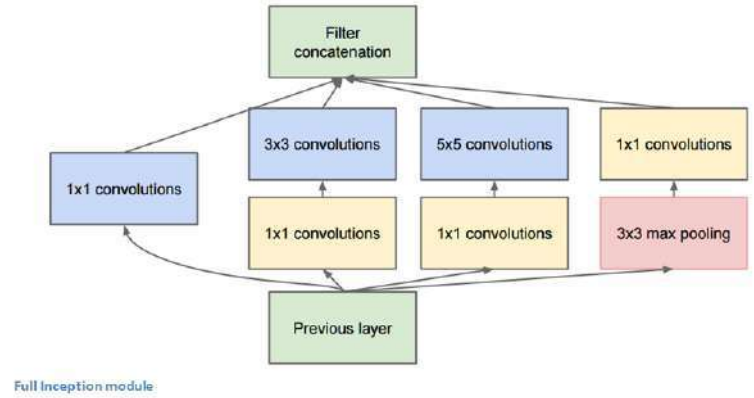

Figura 9.9. Arquitetura do módulo Inception. Figura modificada de [Culurciello 2017]

Podemos destacar como a característica principal desta arquitetura, a melhor utilização dos recursos computacionais dentro da rede. Com um design cuidadosamente elaborado, que aumenta a profundidade e a largura da rede, melhorando o desempenho da rede e diminuir o custo computacional. No ano seguinte, a rede conhecida como Inception-v4 Szegedy et al. 2016, que consistia de alguns melhoramentos da GoogLeNet, alcançou um top-5 erro de 3,08%.

9.2.4.4. VGG Net (2014)

Foi proposto por Simonyan e Zisserman, 2014 um modelo de CNN com até 19 camadas que utilizava filtros 3×3 , obtiveram o segundo lugar do ILSVRC 2014 com taxa de erro de 7,3% [Simonyan e Zisserman 2014]. Os autores concluíram que o modelo generaliza bem uma ampla gama de tarefas e conjuntos de dados, combinando ou superando as tubulações de reconhecimento mais complexas construídas em torno de representações de imagem menos profundas. Os resultados confirmaram a importância da profundidade nas representações visuais.

Foram testadas 6 arquiteturas diferentes, especificadas na Figura 9.10 , e a arquitetura que apresentou o melhor desempenho foi a de rótulo D, que possuía 13 camadas convolucionais, 5 de max-pooling e 3 totalmente conectadas.

9.2.4.5. Microsoft e ResNet (2016)

A rede ResNet proposta por [He et al. 2016], com 152 camadas era a rede mais profunda já treinada no ImageNet. O sistema de rede residual profunda foi 8x vezes maior em profundidade que o VGG Net, utilizou o princípio de "aprendizagem residual" para orientar os projetos de arquitetura de rede.

O desempenho dessa rede foi superior ao de seres humanos, conforme suas habilidades e área de conhecimento, normalmente obtém top-5 erro entre 5 e 10%. Tal façanha

Configuração da ConvNet					
A	A-LRN	B	C	D	E
11 camadas de peso	11 camadas de peso	13 camadas de peso	16 camadas de peso	16 camadas de peso	19 camadas de peso
Entrada (imagem RGB - 224x224)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
softmax					

Figura 9.10. Especificação das 6 arquiteturas da VGG testada, a D se sobressai. Modificada de [Culurciello 2017]

resultou no 1º lugar na competição de classificação ILSVRC na ImageNet com 3,57% de erro 5.

A ideia por trás dos blocos residuais é que uma entrada x passa por uma série de operações de "convolução-relu-convolução". O $F(x)+x$ camada está adicionando, um "novo conhecimento" a entrada x para a $F(x)$ camada. Aqui, $F(x)$ é o residual, como é ilustrado na Figura 9.11.

Numa ResNet, a função $F(x)$ funciona somente como um termo de regularização. Portanto, o espaço de saída é somente uma alteração do espaço de entrada. Os autores desta arquitetura entenderam que os mapas residuais são mais fáceis de serem otimizados por meio dessa alteração no espaço de entrada.

9.3. Experimentos

Um dos primeiros problemas a ser tratado por CNN foi o reconhecimento de dígitos em imagens. Para por em prática os conceitos visto sobre CNN afim de aumentar o aprendizado, utilizaremos uma Rede Neural Convolucional Sequencial de 5 camadas para

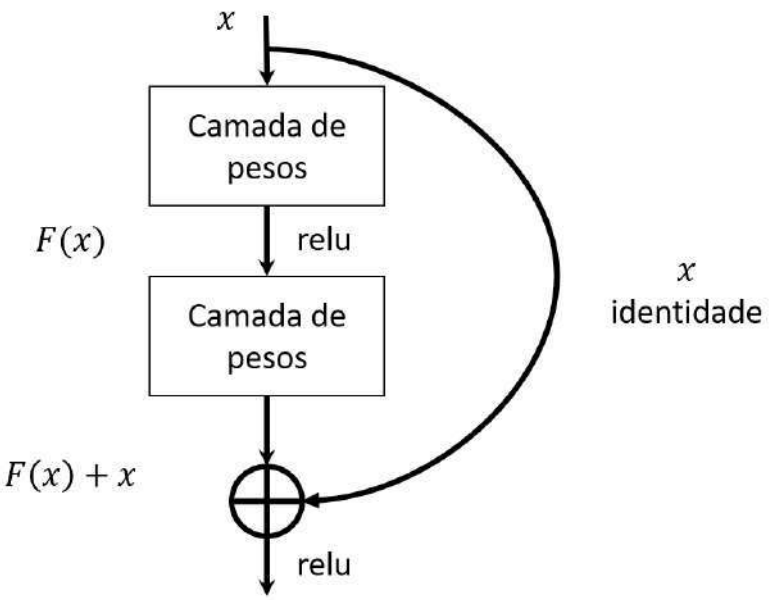

Figura 9.11. Bloco residual. Modificada [Deshpande 2016a]

o reconhecimento de dígitos treinados no conjunto de dados MNIST [Deng 2012]. Para o desenvolvimento escolheu-se construir com a API keras (Tensorflow backend)[ker 2017], que é muito intuitivo. Em primeiro lugar, tem-se que preparar os dados (imagens de dígitos manuscritos), em seguida é modelada, treinada e avaliada o modelo de CNN, para por fim, fazer a predição dos dados de testes a partir dos pesos da rede treinada.

9.3.1. Preparação dos Dados

A primeira parte é a seleção de dados e tratamento para entrada na rede neural. Os dados são lidos do arquivo train.csv que contém os valores dos pixeis de cada imagem juntamente com o rótulo de qual imagem pertence. Os dados são separados em:

- **X_train:** São os valores dos pixeis das imagens, que servem de entrada para a rede neural aprender características e ajustar os pesos com o treinamento a partir desses dados.
- **Y_train:** São os valores dos rótulos das imagens, ou seja, de 0 a 9, que servem de saída desejada para a rede ajustar os pesos com o algoritmo *backpropagation* ao longo das épocas no treinamento.
- **X_test:** São os valores dos pixeis das imagens, mas esses servem para teste, ou seja, quando a rede esta treinada ela utiliza essas matrizes de pixeis para classificar qual rótulo pertence. É utilizado exemplos diferentes dos treinados para testar a capacidade de generalização da rede, o quanto ela consegue classificar amostras diferentes das que foram treinadas.
- **Y_test:** São os valores dos rótulos das imagens, ou seja, de 0 a 9, mas esses também são para teste, os chamadas rótulos verdadeiros, são os rótulos do conjunto de teste

X_test. Ele é utilizado para comparar com a saida da classificação da rede e calcular a acurácia da classificação.

Após a separação para os dados de treinamento e teste, os dados são normalizados, dado que a CNN trabalha melhor com valores no intervalo entre 0 e 1 e os valores dos pixeis estarem entre 0 e 255. Em seguida os dados formatados são salvos para posteriores operações, para diminuir processamento em posteriores operações de treinamento e em seguida teste.

```
#pacotes essencias
import os
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from sklearn.model_selection import train_test_split

np.random.seed(2)


# Preparacao dos dados

# manipulacao de graficos com matploit
sns.set(style='white', context='notebook', palette='deep')

# normaliza os dados ente 0 e 1 visto que a CNN trabalha melhor nesse intervalo
def normalizacao(data):
    return data / 255.0


def create_train_data():

    # le os dados de treino do arquivo
    train = pd.read_csv(os.getcwd()+"/MNIST/data/input/train.csv")

    # rotulos para teinamento, saida desejada
    Y_train = train["label"]

    # deleta os rotulos de train
    X_train = train.drop(labels=["label"], axis=1) # (42000, 784)

    # liberando mais espaco
    del train

    # exibe a quantidade de classes que serao exibidas
    Y_train.value_counts()
    g = sns.countplot(Y_train)
    plt.show()

    #  verificando os valores nulos e ausentes
    X_train.isnull().any().describe()
```

```
45
    # Normaliza os dados
47  X_train = normalizacao(X_train)

49
    # muda a forma em 3 dimensoes
51  X_train = X_train.values.reshape(-1,28,28,1) # (42000, 28, 28, 1)

53
    # Codifica ao de rotulo
55  Y_train = to_categorical(Y_train, num_classes=10)

57  # exibe graficamente a quantidade de imagens que contm cada classe
    g = plt.imshow(X_train[1][:,:,0])
59  plt.show()

61  # seta sementes aleatorias
    random_seed = 2
63
    # divide os dados de treino e validacao para setar no treinamento
65  X_train, X_test, Y_train, Y_test = train_test_split(X_train,
    Y_train, test_size=0.1, random_state=random_seed)

67  # salva os dados de treino e teste
    np.save(os.getcwd() + '/data_npy/X_train.npy', X_train)
69  np.save(os.getcwd() + '/data_npy/X_test.npy', X_test)
    np.save(os.getcwd() + '/data_npy/Y_train.npy', Y_train)
71  np.save(os.getcwd() + '/data_npy/Y_test.npy', Y_test)

73  return X_train, X_test, Y_train, Y_test


75
# carrega os dados de treino a partir de arquivos salvos
77 def load_train_data():
    X_train = np.load(os.getcwd() + '/data_npy/X_train.npy')
79  Y_train = np.load(os.getcwd() + '/data_npy/Y_train.npy')
    return X_train, Y_train
81
def load_test_data():
83  X_test = np.load(os.getcwd() + '/data_npy/X_test.npy')
    Y_test = np.load(os.getcwd() + '/data_npy/Y_test.npy')
85  return X_test, Y_test

87 if __name__ == '__main__':
    create_train_data()
```

Código Fonte 9.1. Preparação dos Dados.

9.3.2. Modelagem e treinamento da CNN

Usamos a API Keras, onde você tem apenas que adicionar uma camada de cada vez, a partir da entrada. A primeira é a camada convolucional (Conv2D). É como um conjunto de filtros aprendizes. Escolheu-se definir 32 filtros para as duas primeiras camadas conv2D

e 64 filtros para as duas últimas. Cada filtro transforma uma parte da imagem (definida pelo tamanho do kernel) usando o filtro do kernel. A matriz do filtro do kernel é aplicada em toda a imagem. Os filtros podem ser vistos como uma transformação da imagem. A CNN pode isolar recursos que são úteis em todos os lugares a partir dessas imagens transformadas (mapas de características).

A segunda camada importante na CNN é a camada de pooling (MaxPool2D). Esta camada simplesmente atua como um filtro de amostragem descendente. Observa os 2 pixels vizinhos e escolhe o valor máximo. Estes são usados para reduzir o custo computacional e, em certa medida, também reduzem a superposição. Nós temos que escolher o tamanho da pooling (ou seja, o tamanho da área agrupado a cada vez) mais a dimensão da pooling é alta, mais importante é o downsampling. Combinou-se camadas convolutivas e de agrupamento, pois a CNN é capaz de combinar recursos locais e aprender mais recursos globais da imagem.

O *Dropout* é um método de regularização, onde uma proporção de nós na camada é ignorada aleatoriamente (definindo seus pontos como zero) para cada amostra de treino. Isso gera aleatoriamente uma proposta da rede e força a rede a aprender recursos de forma distribuída. Esta técnica também melhora a generalização e reduz a superposição. 'RELU' é o retificador (função de ativação max (0, x). A função de ativação do retificador é usada para adicionar não linearidade à rede.

A camada *Flatten* é usada para converter os mapas de recursos finais em um único vetor 1D. Este passo de achatamento é necessário para que você possa usar camadas totalmente conectadas após algumas camadas convolutionais / maxpool. Combina todas as características locais encontradas das camadas convolutivas anteriores. No final, usou-se os recursos em duas camadas (densas) completamente conectadas, que é apenas um classificador de rede neural (ANN). Na última camada (Dense (10, activation = "softmax")), a distribuição de saídas líquidas da probabilidade de cada classe.

```
# pacotes essencias
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from keras.layers import Conv2D, MaxPool2D, Dropout, Flatten, Dense
from keras.models import Sequential
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot as plt
from data import create_train_data

# funcao que criar a arquitetura da rede
def CNN(weights_path=None):

    model = Sequential() # define o model
    # adiciona as camadas de convolucao, max-pooling e Dropout - camada
     exztratora de caracteristicas
    model.add(Conv2D(filters=32, kernel_size=(5, 5), padding='Same',
    activation='relu', input_shape=(28, 28, 1)))
    model.add(Conv2D(filters=32, kernel_size=(5, 5), padding='Same',
    activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
```

```
20      # adiciona as camadas de convolucao, max-pooling e Dropout - camada
         exztratora de caracteristicas
        model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='Same',
       activation='relu'))
22      model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='Same',
       activation='relu'))
        model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
24      model.add(Dropout(0.25))

26      # camadas de classificacao
        model.add(Flatten())
28      model.add(Dense(256, activation="relu"))
        model.add(Dropout(0.5))
30      model.add(Dense(10, activation="softmax"))

32      # condicao para leitura de pesos caso seja passado por parametro
        if weights_path:
34          model.load_weights(weights_path)

36      return model


38

40  if __name__ == '__main__':

42      model = CNN()

44      # Define o optimizador
        optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
46
        # Compila o model
48      model.compile(optimizer=optimizer, loss="categorical_crossentropy",
         metrics=["accuracy"])

50      model_checkpoint = ModelCheckpoint('digitos.hdf5', monitor='loss',
       save_best_only=True)

52      #epochs = 1  # numeros de epocas - quantidade vezes que se repete o
         terinamento
        batch_size = 86 # tamanho do lote - quantidade de imagens a serem
       processadas por vez
54
        # ontendo os dados de treino e teste
56      X_train, X_val, Y_train, Y_val = create_train_data()

58      # gera mais dados para o treinamento
        datagen = ImageDataGenerator(
60          featurewise_center=False,  # set input mean to 0 over the
       dataset
            samplewise_center=False,  # set each sample mean to 0
62          featurewise_std_normalization=False,  # divide inputs by std of
         the dataset
            samplewise_std_normalization=False,  # divide each input by its
         std
64          zca_whitening=False,     # apply ZCA whitening
```

```
        rotation_range=10,       # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range=0.1,          # Randomly zoom image
        width_shift_range=0.1,   # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,   # randomly flip images
        vertical_flip=False)  # randomly flip images
    datagen.fit(X_train)

    # treina o model
    history = model.fit_generator(datagen.flow(X_train, Y_train, batch_size=batch_size), epochs=epochs, validation_data=(X_val, Y_val),
                                  verbose=1, steps_per_epoch=X_train.shape[0] // batch_size, callbacks=[model_checkpoint])

    # exibe por graficos os valores de perda no treinamento
    fig, ax = plt.subplots(2, 1)
    ax[0].plot(history.history['loss'], color='b', label="Training loss")
    ax[0].plot(history.history['val_loss'], color='r', label="validation loss", axes=ax[0])
    legend = ax[0].legend(loc='best', shadow=True)

    ax[1].plot(history.history['acc'], color='b', label="Training accuracy")
    ax[1].plot(history.history['val_acc'], color='r', label="Validation accuracy")
    legend = ax[1].legend(loc='best', shadow=True)
    plt.show()
```

Código Fonte 9.2. modeladagrm, treinamento e avalição da CNN

9.3.3. Predição

Após o treinamento, o aprendizado da CNN é salvo os pesos, para ser possível realizar a predição apenas carregando os pesos sem a necessidade de treinar novamente para realizar a classificação. Para realizar a predição, primeiramente é carregado os dados de teste os quais já estão normalizados. Em seguida é carregado o modelo da CNN pelos pesos salvos no treinamento. Após essas etapas a CNN faz a predição sobre os dados de teste classificando entre os dígitos de 0 a 9, para que, com o resultado da classificação, seja confrontado com o verdadeiro rótulo da imagem testada para calcular métricas estatísticas como acurácia.

```
from keras.optimizers import RMSprop
from sklearn.metrics import confusion_matrix
from data import load_test_data
from train import CNN
import numpy as np
from MNIST.functions import plot_confusion_matrix
from util import main_validacao
from sklearn.metrics import accuracy_score
```

```
9
def load_model_network():
11
    # carrega os dados de teste para a predicao
13  X_test, Y_test = load_test_data()

15  #carrega o modelo e os pesos ja treinados
    model = CNN('digitos.hdf5')
17  optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy",
     metrics=["accuracy"])
19
    # faz a predicao sobre as imagnes de teste
21  Y_pred = model.predict(X_test)

23  # Converte as classes das predicoes para valores de 0 a 9
    Y_pred_classes = np.argmax(Y_pred, axis=1)
25
    # Converte os classes corretas das predicoes para valores de 0 a 9
27  Y_true = np.argmax(Y_test, axis=1)

29  # calcula a matriz de confusao
    confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
31
    # exibe a acuracia sobre a predicao realizada
33  print(" ACURCIA = %.2f" %(accuracy_score(Y_true, Y_pred_classes)
    *100), "%")

35  # exibe graficamente a matriz de confusao
    plot_confusion_matrix(confusion_mtx, classes=range(10))
37
39
41
43
if __name__ == '__main__':
45  """"""
    load_model_network()
```

Código Fonte 9.3. predição dos dados de testes

Por fim, para melhor análise dos dados é gerada graficamente a matriz de confusão o qual demostra a predição feita pela CNN. A Figura 9.12 ilustra a matriz de confusão gerada pela classificação sobre o conjunto de dados de Teste. As linhas representam os rótolos verdeiros sobre as imagens testadas, e as colunas representam os rótulos classificados pela CNN, mas é na diagonal que é representado a quantidade de predições feitas pela CNN corretas. Na primeira linha, de 415 imagens do digito 0 testadas, a CNN classificou corretamente 409 como corretas, 1 como o dígito 2, duas como 3, duas como 6, e uma como 9.

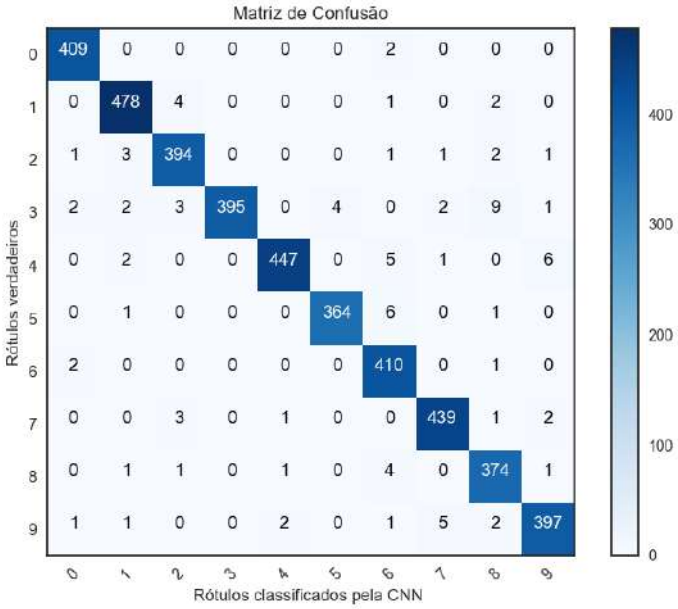

Figura 9.12. Matriz de Confusão.

9.4. Considerações Finais

As abordagens que usam CNNs para tarefas de classificação e detecção de objetos em imagens se sobressaem no que diz respeito a Visão Computacional. Diversas aplicações vem utilizando CNNs, após o ganho de popularidade em 2012 com a competição de reconhecimento de objetos, a *ImageNet*. O objetivo do curso é desenvolver uma compreensão clara sobre a motivação para as CNNs e *Deep Learning*. Conceitos inicias permitem uma assimilição da teoria com aplicações, além de noções de como projetar um código ou sistemas a partir do que foi visto.

É importante que o aluno não se limite apenas as CNNs no que diz respeito a Deep Learning mas, que explore outros algoritmos e arquiteturas de redes como, por exemplo, Redes Autocodificadoras e Redes Neurais Recorrentes, além de técnicas para treinamento de redes profundas.

Referências

[ker 2017] (2017). Keras: The python deep learning library.

[Araújo et al.] Araújo, F. H., Carneiro, A. C., e Silva, R. R. Redes neurais convolucionais com tensorflow: Teoria e prática.

[Beam 2017] Beam, A. L. (2017). Deep learning 101 - part 1: History and background. https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html.

[Culurciello 2017] Culurciello, E. (2017). Neural network architectures. https://medium.com/towards-data-science/neural-network-architectures-156e5bad51ba.

[Deng 2012] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142.

[Deshpande 2016a] Deshpande, A. (2016a). The 9 deep learning papers you need to know about (understanding cnns part 3). https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html.

[Deshpande 2016b] Deshpande, A. (2016b). A beginner's guide to understanding convolutional neural networks part 2. https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/.

[Ferreira 2017] Ferreira, A. d. S. (2017). Redes neurais convolucionais profundas na detecção de plantas daninhas em lavoura de soja. Master's thesis.

[Haykin 2001] Haykin, S. (2001). *Redes neurais: princípios e prática*. Bookman Editora.

[He et al. 2016] He, K., Zhang, X., Ren, S., e Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hinton et al. 2006] Hinton, G. E., Osindero, S., e Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

[Kim 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

[Krizhevsky et al. 2012] Krizhevsky, A., Sutskever, I., e Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[LeCun et al. 1990] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., e Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

[LeCun et al. 2010] LeCun, Y., Kavukcuoglu, K., e Farabet, C. (2010). Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256. IEEE.

[Libralão et al. 2003] Libralão, G. L., Oshiro, R. M., Netto, A. V., Carvalho, A., e Oliveira, M. (2003). Técnicas de aprendizado de máquina para análise de imagens oftalmológicas. *São Paulo. Universidade de São Paulo*.

[Morais 2017] Morais, L. (2017). Introdução Às redes neurais artificiais. http://www.lmtech.info/index.php/tecnologia/inteligencia-artificial/129-introducao-a-redes-neurais-artificiais.