REDES NEURAIS DEEP LEARNING COM TENSORFLOW

FALCÃO, João V. R.[1]
MOREIRA, Vinicius de A.[1]
SANTOS, Flávia A. O.[2]
RAMOS, Celso de A. [2]
[1] Acadêmico de Ciência da Computação, Universidade José do Rosário Vellano
[2] Professor(a) de Ciência da Computação, Universidade José do Rosário Vellano, ORCID

Resumo

O objetivo deste trabalho é realizar um estudo dos algoritmos de redes neurais de aprendizado profundo implementados em TensorFlow. Foram apresentados conceitos básicos fundamentais de *Deep Learning* e suas aplicações. Além disso, foi apresentada uma descrição do *framework* TensorFlow como ferramenta de implementação das redes neurais, abordando seu funcionamento e ciclo de vida. Foi implementada uma aplicação prática para reconhecimento de partes de vestuário com *deep learning*. Como conclusão, destaca-se as áreas de aplicação das Redes Neurais de Aprendizado Profundo e como elas podem auxiliar na solução de problemas do mundo real, além de reflexões sobre o futuro da inteligência artificial.

Palavras-Chave. *Aprendizado Profundo, Redes Neurais, Reconhecimento de Padrões, TensorFlow, Aprendizado de Máquina.*

Abstract

The main purpose of this paper is the study of deep learning neural network algorithms, with implementation in TensorFlow. It will cover fundamental concepts of Deep Learning and its applications. The Tensorflow framework will be presented as a tool for the implementation of neural networks, addressing their functioning and life cycle. A practical application for the recognition of garments using a deep learning is implemented. It concludes by highlighting application areas of the Deep Learning Neural Networks and how they can help us solve real-world problems, as well as reflections on the future of artificial intelligence.

Keywords. *Deep Learning, Neural Networks, Pattern Recognition, TensorFlow, Machine learning.*

1 Introdução

Inteligência Artificial e *Machine Learning* já não são mais assuntos apenas para ficção científica. As discussões sobre máquinas inteligentes capazes de superar as habilidades humanas foram feitas antes mesmo dos primeiros computadores. A Inteligência Artificial (I.A) começa a ser reconhecida como Ciência a partir de 1956, impulsionando novos trabalhos científicos na área. Ainda que no período sendo apenas teórica, a I.A chamava atenção de matemáticos e teóricos da computação da época. Por muitos anos, o principal limitante dos algoritmos inteligentes foi o poder computacional, limitação essa que vem sendo rompida nos últimos anos. Com a evolução do hardware e do software, houve uma explosão no que se refere ao uso de algoritmos inteligentes, tais como algoritmos de recomendação baseado em redes neurais, classificação de imagens, reconhecimento facial, reconhecimento de íris, entre outros. Os *smartphones* estão repletos de algoritmos treinados e calibrados de acordo com as necessidades dos usuários. Esses algoritmos operam para dar uma imersão maior no uso das aplicações, criando uma sensação de conforto, onde tudo está ao seu alcance.

Uma das principais características desse tipo de algoritmo inteligente, é a capacidade de aprender com a experiência e, depois de calibrado, ser capaz de classificar objetos do mundo real de forma automática a precisa. O Google já utiliza algoritmos de classificação de imagem nos seus aplicativos, como por exemplo Google Fotos, onde o próprio aplicativo toma a decisão de como organizar suas fotos em álbuns de acordo com a imagem. Assim, suponha que foram tiradas várias fotos de um aniversário, essas fotos irão para o Google Fotos e ficariam misturadas com todas as demais fotos. Automaticamente, o algoritmo inteligente do aplicativo irá analisar as fotos e organizá-las em um álbum “Aniversário”, por exemplo. Isso é a classificação automática de uma rede neural treinada para classificar fotos de aniversários.

A análise e classificação de imagens representa um ponto de apoio a diversas áreas, como na medicina, na indústria, nos carros autônomos, etc. A diversidade de aplicações de *Deep Learning*, está diretamente associada à análise da informação citada acima. Podemos analisar diversos tipos de dados, que vão desde imagens comuns até imagens acústicas, sísmicas, de satélites, infravermelhas, magnéticas, etc.

Este trabalho permeia os fundamentos de Redes Neurais Artificiais, bem como sua evolução. As *Deep Learning* têm como objetivo ser um guia introdutório para criação de Redes Neurais Artificiais de Aprendizado Profundo através da ferramenta TensorFlow, desenvolvida pela Google. O objetivo é desenvolver uma aplicação de base para classificação de peças de roupas dentre 10 categorias diferentes.

2 Redes Neurais Artificiais

Redes neurais artificiais são sistemas de computação vagamente inspirados pelas redes neurais biológicas que constituem os cérebros animais. A rede neural em si não é um algoritmo, mas a estrutura de muitos algoritmos diferentes de aprendizado de máquina para trabalhar juntos e processar entradas de dados complexas. Tais sistemas aprendem a executar tarefas considerando exemplos, sem serem programados com regras específicas de tarefas. Por exemplo, no reconhecimento de imagem, as redes neurais podem aprender a identificar imagens que contenham gatos analisando exemplos que tenham sido rotulados manualmente como "gato" ou "não gato" e, em seguida, usar os resultados para identificar gatos em outras imagens. As redes neurais fazem isso sem qualquer conhecimento prévio sobre gatos, por exemplo, que eles têm pele, rabo, bigode e cara de gato. Em vez disso, eles identificam automaticamente as características do material de aprendizagem que processam (WALKARN, 2019).

2.1 Neurônio

O neurônio artificial é o componente de construção das Redes Neurais Artificiais projetados para simular a função de um neurônio biológico. Os sinais que chegam, chamados entradas, multiplicadas pelos pesos de conexão são primeiro somados e depois passam por uma função de transferência para produzir a

saída para esse neurônio. A função de ativação é a soma ponderada das entradas do neurônio e a função de transferência mais comumente usada é a função sigmóide.

2.2 Backpropagation

O algoritmo de *backpropagation* foi originalmente introduzido na década de 1970, mas sua importância não foi totalmente apreciada até um famoso artigo de 1986 de David Rumelhart, Geoffrey Hinton e Ronald Williams. Hoje o algoritmo tornou-se líder em utilização para solucionar o treinamento e atualização dos pesos das redes neurais. Existem outras soluções que vêm sendo abordadas, como por exemplo, o uso de Algoritmos Genéticos para atualização de pesos como alternativa ao *backpropagation*.

No coração do *backpropagation*, está uma expressão para a derivada parcial $\partial C / \partial w$ da função de custo C em relação a qualquer peso w (ou *bias* b) na rede. A expressão informa a rapidez com que o custo muda quando os pesos e os *biases* são alterados. O *backpropagation* não é apenas um algoritmo rápido para aprender. Na verdade, ele fornece *insights* detalhados sobre como alterar os pesos e como os *biases* altera o comportamento geral da rede. Em linhas gerais, o algoritmo *backpropagation* é a base para criação de uma rede neural capaz de aprender. O *framework* TensorFlow faz uso desse algoritmo em sua implementação (NIELSEN, 2019).

3 Framework TensorFlow

Criado pela equipe do Google Brain, TensorFlow é um framework *open-soure* desenvolvido para Python e JavaScript, que auxilia no desenvolvimento de soluções com *machine learning*. Pode ser executado sobre diversas plataformas e arquiteturas, incluido CPUs, GPUs e as recentes TPUs (*Tensor Processing Unit*). Atualmente, é um dos principais *frameworks* do mercado para criação de redes neurais *deep learning*. Com ele, é possível agilizar e facilitar o processo de obtenção de dados, treinar modelos, realizar *predictions* e refinar resultados futuros (TENSORFLOW, 2019).

3.1 Arquitetura do TensorFlow

TensorFlow utiliza uma API rica em linguagem Python, trazendo um desenvolvimento simplificado para o programador final. Sua execução se passa sobre uma aplicação de alta performance escrita em C/C++. A equipe de engenheiros da Google, uniu as facilidades do Python com a alta performance do C/C++. Sua arquitetura se divide em três partes principais, sendo elas (ARAUJO et al. 2017):

- Pré-processamento dos dados;
- Construção dos modelos;
- Treinamento e estimativas do modelo criado.

O modelo estrutural do framework é ilustrado na FIGURA 1. A entrada é um *array* multidimensional, chamado de **tensors**. São construídos **fluxogramas** (*flow charts*) das operações que serão realizadas sobre a entrada. Os dados de entrada entram em uma extremidade, fluem por um número determinado de operações e saem na outra extremidade. Esse comportamento caracteriza o nome TensorFlow do *framework* (TENSORFLOW, 2019)..

3.2 Componentes do TensorFlow

3.2.1 Tensor

O nome TensorFlow é derivado diretamente da sua ideia central: o **Tensor**. No *framework*, todas as operações de computação envolvem tensors. Um tensor é um **vetor** ou **matriz** de N dimensões que

representa todos os tipos de dados. Todos os valores em um tensor mantêm um tipo de dados idêntico com uma forma (*shape*) conhecida (ou parcialmente conhecida). A forma dos dados é a dimensionalidade da matriz.

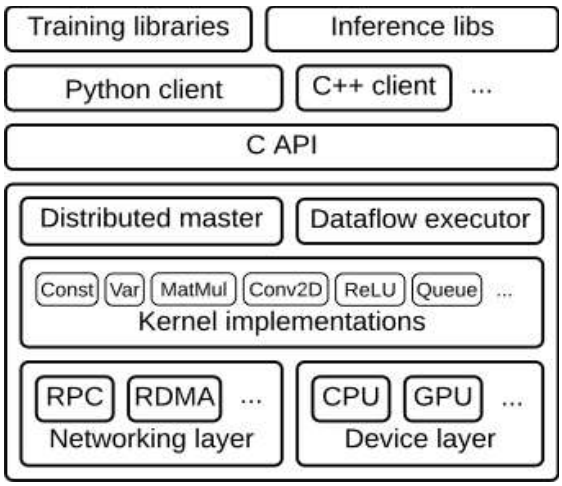

FIGURA 1. Modelo arquitetural do TensorFlow.

Um tensor pode ser originado a partir de uma entrada de dados ou o resultado de uma operação. Todas as operações são conduzidas dentro de um **gráfico** (*graph*). Um gráfico é um conjunto de operações realizadas de formas sucessivas. Cada operação é chamada de "*Op Node*" e são conectadas umas as outras.

3.2.2 Graph

TensorFlow faz uso de um *framework* específico para gerenciar os gráficos. O gráfico, como explicado anteriormente, reúne um conjunto de operações realizadas de forma sequencial, agregando e descrevendo todos os cálculos de séries feitos durante o treinamento. Existem diversas vantagens em utilizar os *graphs*, entre elas (YEGULALP, 2019):

- Foi criado para ser executado em várias arquiteturas, dentre elas, CPUs, GPUs e até mesmo em sistemas operacionais *mobile*;
- O gráfico possui portabilidade, o que significa que é possível preservar os cálculos para uso imediato ou posterior. O gráfico pode ser salvo para ser executado no futuro;
- Todos os cálculos no gráfico são feitos conectando os **tensors** em conjunto.

3.2.3 Keras API

De acordo com a documentação oficial do TensorFlow, "Keras é uma API de alto-nível (*high-level*) para construir e treinar modelos de aprendizado profundo. É usado prototipagem rápida, pesquisa avançada e produção, com 3 vantagens chave: *User friendly, Modular and composable, Easy to extend.*". O Keras possui uma interface simples, pensada para ser fácil e de uso comum. Possui um mecanismo de *feedback* de erros, facilitando o desenvolvimento por parte do programador final. Além disso, os modelos são constituídos interligando blocos configuráveis, com baixa restrição, permitindo criar modelos complexos sem que isso se torne um grande conglomerado de código para o desenvolvedor (KERAS, 2019).

A API provê uma variedade de funções de ativação e métricas que podem ser usadas para otimizar as redes neurais de aprendizado profundo, além de permitir que seja feito medições quanto a performance e precisão dos modelos criados.

3.3 Benefícios do TensorFlow

O maior benefício que o TensorFlow oferece para o desenvolvimento de sistemas inteligentes é a abstração. Em vez de lidar com os detalhes básicos da implementação de algoritmos ou de descobrir formas adequadas de ligar a saída de uma função à entrada de outra, o desenvolvedor pode se concentrar na lógica geral da aplicação ou no problema que se deseja resolver (TENSORFLOW, 2019).

O TensorFlow provê diversas ferramentas de depuração e introspecção das aplicações de *machine learning*, o que permite avaliar e modificar cada operação dos gráficos separadamente e de forma explícita, em vez de construir o gráfico por inteiro e tentar avaliar tudo de uma vez.

Além das facilidades de desenvolvimento, TensorFlow torna-se uma ferramenta confiável, uma vez que é um dos carros-chefes da área de Inteligência Artificial da Google. A Google não só impulsionou o desenvolvimento do *framework*, como incentivou seu uso criando ofertas de servidores dedicados com GPUs e até mesmo as poderosas TPUs, que possuem um desempenho acelerado na nuvem do Google. É possível desenvolver soluções em nuvem, escaláveis e com alta disponibilidade na web.

4 Redes Neurais de Aprendizado Profundo

Redes Neurais de Aprendizado Profundo são basicamente redes neurais multicamadas onde o número de camadas ocultas é maior que 1. São amplamente utilizadas na implementação de sistemas inteligentes e possuem uma capacidade de aprendizado ampliado em relação a Redes Neurais Artificiais comuns. Além do maior número de camadas, esse tipo de Rede Neural *Deep Learning* também precisa de algumas adaptações nos algoritmos de atualização dos pesos e alguns parâmetros a mais que serão discutidos ao longo deste artigo.

Praticamente, todos os sistemas que utilizam inteligência artificial atualmente fazem uso de *Deep Learning*, visto que existe poder computacional para utilização dessa tecnologia. Antes, a *Deep Learning* era limitada pela capacidade dos computadores e servidores que executam milhões de cálculos matemáticos para realizar os passos de execução, avaliação e atualização dos pesos que estão presentes em todas as redes neurais (ARAUJO et al, 2017).

4.2 Bias

O bias é um parâmetro constante da rede neural que é inserido junto às entradas. Ele é tratado como uma entrada e também possui pesos, que podem ser atualizados pelo algoritmo de atualização de pesos que estiver sendo utilizado. Bias, em sua tradução literal, significa “viés” ou “preconceito”, trata-se de uma forma de direcionar o aprendizado da rede neural e também impedir que ela transmita valores zerados por acidente, fazendo com que a rede neural caia em mínimos locais e não consigam aprender corretamente. Na FIGURA 2, é representado o parâmetro bias em atuação sobre um neurônio único.

4.2 Funções de Ativação

A principal finalidade das funções de ativação é converter um sinal de entrada de um neurônio em uma rede neural artificial para um sinal de saída. Essa saída será usada na próxima camada. Especificamente, em uma rede neural artificial, é realizada a soma dos produtos das entradas (X) e de seus pesos correspondentes (W). Em seguida, é aplicada uma função de Ativação f(x) para obter a saída dessa camada e

alimentá-la como uma entrada para a próxima camada. Neste trabalho, foi utilizada as funções de ReLU e Sigmoid (ARAUJO et al, 2017).

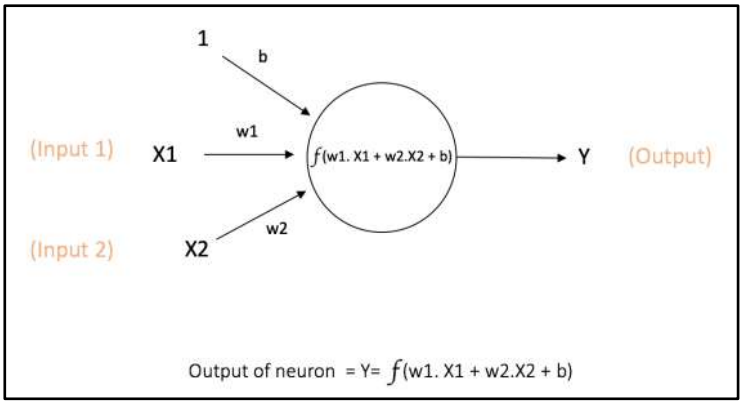

FIGURA 2. Parâmetro bias em uso. (Foto: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/)

4.2.1 *Step function*

A primeira função de ativação que foi usada para redes neurais artificiais é chamada de *step function*, criada por Oliver Heaviside (1850–1925). Ela funciona basicamente como se fosse um degrau. A *step function* é baseada em um *threshold*, se for maior que um valor definido ela é ativada, caso contrário, ela fica desativada.

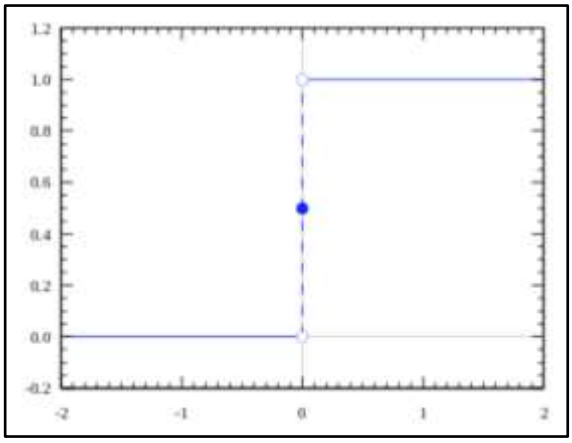

FIGURA 3. Gráfico da função de passo.

Um dos problemas da utilização da *step function* na implementação de redes neurais é a mudança brusca de estado. Uma vez que uma pequena alteração na rede neural causará um efeito muito maior do que o esperado, já que a função trabalha apenas com 0 e 1. Para isso, existem outras funções de ativação que são mais comumente utilizadas.

4.2.2 Função Sigmóide

A curva da Função Sigmóide (FIGURA 4) se parece com uma forma em S. Ela foi criada para resolver os problemas da *step function*, sendo uma função de passo suavizada. Quaisquer alterações nos valores de X nessa região, entre X -2 a X +2, farão com que os valores de Y sejam alterados significativamente. Isso significa que essa função tem a tendência de trazer os valores de Y para qualquer extremidade da curva.

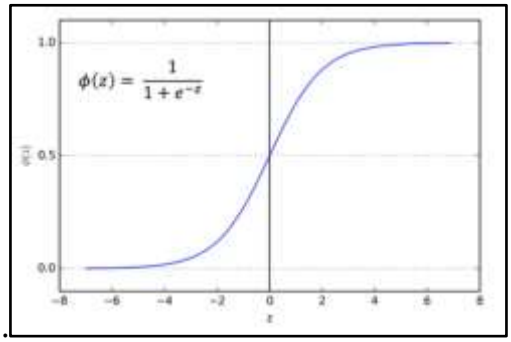

FIGURA 4. Gráfico da função Sigmóide.

O principal benefício de utilizar a função sigmóide é a obtenção de uma certa suavidade nas alterações dos valores da rede neural. Não existe um resultado abrupto, uma vez que ela trabalha em um intervalo entre 0 e 1, e não absolutamente 0 ou 1.

4.2.2 Função ReLU

ReLU é a sigla para *Rectified Linear Unit*, sendo a função de ativação mais usada em modelos de aprendizagem profunda. Conforme ilustrado na FIGURA 5, a função retorna 0 se receber qualquer entrada negativa, mas para qualquer valor positivo x, ela retorna um valor. Então, pode ser escrito como f (x) = max (0, x). ReLU é convencionalmente usada como uma função de ativação para as camadas ocultas em uma rede neural profunda e não na camada de saída, pois ela cresce os valores infinitamente. Por isso, neste trabalho, a ReLu foi usada para as camadas ocultas, sendo a função sigmoidal utilizada na camada de saída assim para que os valores sejam suavizados.

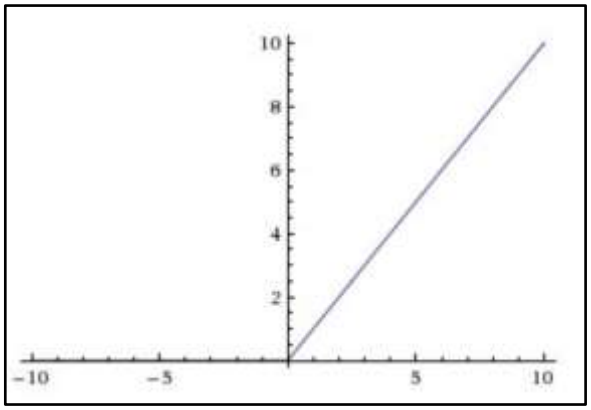

FIGURA 5. Gráfico da função ReLU.

4.3 Adam Optimizer Algorithm

Adam é um algoritmo de otimização que pode ser usado em vez do procedimento clássico de descida de gradiente estocástico para atualizar os pesos da rede de forma iterativa com base nos dados de treinamento. O método é simples de implementar e é computacionalmente eficiente, tem poucos requisitos de memória, é invariante ao reescalonamento diagonal dos gradientes e é bem adequado para problemas que são grandes em termos de dados e / ou parâmetros.

5 Reconhecimento de Partes de Vestuário

A ideia deste capítulo é construir uma rede neural capaz de classificar uma peça de roupa dentre 10 categorias disponíveis: Camiseta, Calça, Moletom, Vestido, Casaco, Sandália, Camisa, Sapatilha, Bolsa, Bota. Para isso, foram utilizadas técnicas dilucidadas ao decorrer do artigo, bem como o *framework* TensorFlow para seu desenvolvimento.

5.2 Base de dados

A base de dados utilizada vem do conjunto de *datasets* da API do Keras, Na documentação oficial (https://keras.io/datasets/), é possível encontrar pelo menos 7 *datasets* com mais de 50 mil registros cada um, possibilitando realizar testes e treinar redes neurais com objetivo de estudar seu funcionamento. Neste trabalho, foi utilizado o *dataset Fashion-MNIST database of fashion articles*, que possui um total de 60,000 imagens de peças de roupas rotuladas e prontas para uso.

5.3 Desenvolvimento

O primeiro passo é importar as dependências. As duas principais são as bibliotecas do TensorFlow e do Keras, como mostrado na linha 2 e 3 do código da FIGURA 6. A *library* "tensorflow", nomeada como "tf", provê acesso às funções de ativação. A biblioteca Keras permite a criação dos modelos e o uso da base de dados.

```
1   # TensorFlow and tf.keras
2   import tensorflow as tf
3   from tensorflow import keras
4
5   # Helper libraries
6   import matplotlib.pyplot as plt
7   import HelperFunctions as hf
```

FIGURA 6. Bibliotecas usadas no desenvolvimento..

Após o *import*, é necessário recuperar os dados da base de dados do Keras. Para isso, é utilizado o método *load_data*() da propriedade "*fashion_mnist*", que irá carregar todo o *dataset* para dentro das variáveis *train_images*, *train_labels*, *test_images*, *test_labels*. Nota-se que duas variáveis são nomeadas com "*train*" no início, e outras duas com "*test*". Isso acontece, pois, a biblioteca do Keras provê uma base de dados para usar durante o treinamento, e ainda um conjunto de 10 mil itens para usar nos testes e avaliar se a rede neural está bem calibrada. A FIGURA 7 exemplifica como é feito o carregamento dos dados a partir do Keras.

```
9   fashion_mnist = keras.datasets.fashion_mnist
10
11  # Buscando dataset da base de dados do tensorflow
12  (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

FIGURA 7. Recuperando dataset usado para treinar a rede neural.

A etapa mais simples é criar os rótulos. Para a rede neural, tudo são números e funções matemáticas. Mas para ser legível para o ser humano, é necessário que a informação seja expressada em forma de texto. A

ideia é que a rede neural retorne qual índice do *array* "*class_names*" que ela categorizou a entrada informada e, a partir disso, é buscado o rótulo no *array* apresentado na FIGURA 8.

```
14    # Classes de roupas/ sapatos que a rede neural deve reconhecer
15    class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
16                   'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

FIGURA 8. Criando array de categorias de roupas.

Para o TensorFlow, os números estão entre 0 e 1, sendo a saída um número entre 0 e 1. As funções de ativação irão sempre normalizar os valores para esse intervalo pequeno, evitando *overflows* por conta de números muito grandes. Na FIGURA 9, é feito uma conversão dos *datasets* de treino e teste para que os valores dos *pixels* das imagens fiquem dentro do intervalo requerido.

```
18    # Essa linha transforma todos os pixels das imagens em
19    # valores entre 0.0 e 1.0 sem perder a característica
20    train_images = train_images / 255.0
21    test_images = test_images / 255.0
```

FIGURA 9. Conversão dos valores das imagens para valores entre 0 e 1.

Finalmente, uma das etapas mais importantes é a definição do modelo que será utilizado. Neste caso, devem ser considerados alguns conceitos sobre os tipos de camadas que a API do Keras disponibiliza e a forma de montar o modelo. O primeiro item, mostrado na FIGURA 9, é o método "Sequential". Segundo a documentação oficial do Keras, o método Sequential cria uma estrutura linear e em pilha das camadas da rede neural. Isso definirá a forma da rede neural. Em relação às camadas, nota-se que a primeira *layer* (camada) é criada pelo método *keras.layers.Flatten*. Uma camada Flatten é comumente usada para ser o *input*. Isso se deve pelo fato de ser uma camada que não altera nenhum valor da entrada, uma vez que não possui função de ativação. No restante da rede neural, são criadas 3 camadas ocultas e mais uma de saída, todas através do método *keras.layers.Dense*. Uma camada densa, como diz a documentação do Keras, é simplesmente uma camada densamente conectada com as outras, significando que cada neurônio da camada terá uma ligação com os neurônios da camada anterior e da camada subsequente, além de implementar a operação de ativação e também o parâmetro bias.

Na camada Flatten, definimos um *input_shape* de 28x28, que é a resolução das imagens do *dataset*. A resolução é para que não seja um processamento gigantesco trabalhar com essa rede neural e realizar medições e testes. Na camada Dense, definiu-se no primeiro parâmetro, a quantidade de neurônios da camada. No exemplo da FIGURA 10, foram utilizados 128 neurônios nas camadas ocultas. No segundo parâmetro, foi definida a função de ativação, nesse caso, a função ReLU (Rectified Linear Units). Na última camada, a de saída, foram utilizados 10 neurônios, onde cada neurônio representa uma saída esperada para o vetor *class_names* criado anteriormente. A rede neural utilizada na camada de saída é a função de ativação SoftMax, que trata-se de uma função de normalização de um exponencial. Essa função de ativação foi utilizada pelo fato da função ReLU não resultar em valores entre 0 e 1. Desse modo, foi normalizada a saída para identificação da decisão tomada pela rede neural.

Considerando os passos anteriores, foi definida uma rede neural com 3 camadas ocultas, 1 camada de entrada e 1 camada de saída. Na camada de entrada, tem-se 784 neurônios que representam cada pixel da imagem de dimensão 28x28. Nas camadas ocultas, foram utilizados 128 neurônios em cada uma das 3

camadas. Na camada de saída, foram utilizados 10 neurônios, representando cada categoria do vetor "*class_names*".

```
23  # Criando modelo da rede neural com:
24  # - 1 camada de entrada com (28 * 28) neurônios
25  # - 3 camadas escondidas com 128 neurônios cada
26  # - 1 camada de saída com 10 neurônios que representam
27  #   a posição no vetor de class names
28  model = keras.Sequential([
29      keras.layers.Flatten(input_shape=(28, 28)),
30      keras.layers.Dense(128, activation=tf.nn.relu),
31      keras.layers.Dense(128, activation=tf.nn.relu),
32      keras.layers.Dense(128, activation=tf.nn.relu),
33      keras.layers.Dense(10, activation=tf.nn.sigmoid)
34  ])
```

FIGURA 10. Definição do modelo através do Keras API.

Não basta ter a estrutura, modelos e dados carregados, é necessário compilar a rede neural, passando um algoritmo de otimização. Foi utilizado o *Adam Optimizer Algorithm*, um algoritmo de otimização estocástico, baseado em gradiente. Também é nesta etapa que são definidas as métricas para avaliar o desempenho da rede neural. Neste caso, foi definida uma função para cálculo de *loss* (perda), chamada "*Sparse Categorical Crossentropy*", que irá informar o quanto ainda há para aprender sobre a base de dados. Essa estimativa define se é necessário continuar o treinamento ou se a rede já aprendeu o máximo possível com os dados processados. A única métrica usada é a "*Accuracy*" ou "Precisão", que é o quanto a rede neural está sendo assertiva nos resultados. Na FIGURA 11 é ilustrado como foi feita a compilação da rede neural.

```
41  model.compile(optimizer=tf.train.AdamOptimizer(),
42                loss='sparse_categorical_crossentropy',
43                metrics=['accuracy'])
```

FIGURA 11. Compilando, otimizando e definindo métricas do modelo..

Na FIGURA 12, é apresentado o trecho de código necessário para realizar o treinamento da rede neural. O TensorFlow resume tudo em uma única linha, através do método *model.fit()*. Esse método é responsável por realizar o treinamento dos modelos criados nos passos anteriores. São passados os *datasets* de treinamento e as *labels* para a função, assim como a quantidade de épocas que o algoritmo deverá treinar.

Também foi feita uma avaliação após o treinamento dos resultados da rede neural, através do método *model.evaluate()*, passando o *dataset* de teste e as *labels* de teste. Através desse método, o TensorFlow retornará os dados sobre *loss* e precisão da rede neural, sendo estas as métricas definidas anteriormente durante a configuração dos nossos modelos.

```
45  # Esta é a etapa do treino, é aqui onde passamos o dataset
46  # para treinar e as labels esperadas
47  model.fit(train_images, train_labels, epochs = 5)
48
49  # Recuperando a taxa de loss e a precisão do nosso modelo
50  # através de um outro dataset para teste
51  test_loss, test_acc = model.evaluate(test_images, test_labels)
52
53  print('Test accuracy:', test_acc)
54  print('Test loss:', test_loss)
```

FIGURA 12. Treinamento e avaliação do modelo..

5.4 Resultados e Discussões

Após o treinamento, a rede neural estará pronta para fazer suas classificações e predições. O modelo que foi estabelecido, quantidade de épocas e *dataset* utilizado trouxeram resultados entre 95% e 99% de precisão, em alguns casos chegando até 100% de certeza. O código ilustrado na FIGURA 13 exemplifica a obtenção e exibição dos resultados.

```
56  # Recuperando as predictions para as
57  # imagens do dataset de teste
58  predictions = model.predict(test_images)
59
60  # Montando gráficos e exibindo resultados
61  imageIndexToShow = 0
62
63  plt.figure(figsize=(6,3))
64
65  plt.subplot(1,2,1)
66
67  hf.plot_image(imageIndexToShow, predictions, test_labels, test_images, class_names)
68
69  plt.subplot(1,2,2)
70
71  hf.plot_value_array(imageIndexToShow, predictions, test_labels)
72
73  plt.show()
```

FIGURA 13. Predição e exibição de resultados da rede neural.

Alternando o parâmetro "*imageIndexToShow*", é possível visualizar o resultado de diferentes imagens do *dataset* de teste, conforme ilustado na FIGURA.

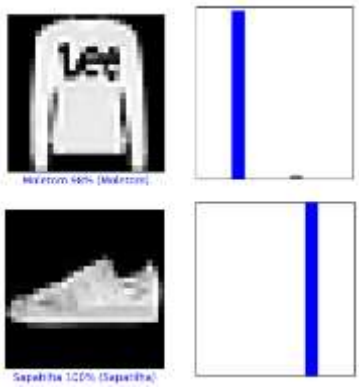

FIGURA 14. Resultados da classificação.

6 Conclusão

As Redes Neurais de Aprendizado Profundo abrem um leque de possibilidades para solucionar problemas que antes não eram possíveis. Vários problemas onde era necessária a intervenção humana, estão sendo solucionados através de algoritmos inteligentes, que aprendem e se adaptam. Atualmente é difícil imaginar um mundo onde a I.A. não está presente. A Inteligência Artificial é utilizada em vários cenários, como já citado neste artigo, em carros autônomos, medicina, indústria e experiência do usuário. É importante salientar que as aplicações atuais são possíveis apenas por conta das redes Deep Learning, uma vez que elas têm uma capacidade de aprendizado maior, gerando resultados mais precisos. A informação propagada pela rede neural é mais concisa e confiável, uma vez que temos múltiplas camadas atuando no resultado. É um modelo que se aproxima mais do cérebro humano.

Quanto ao framework TensorFlow, a Google vem o mantendo e desenvolvendo novas versões que abrangem não só a plataforma Python para servidor, mas recentemente, lançou seu *framework* TensorFlow.JS, que provê o framework em *client-side* sobre linguagem JavaScript, provendo um maior

poder e adaptabilidade do *framework* para qualquer plataforma. É possível esperar mais ainda da plataforma, uma vez que a mesma é o ponto-chave de boa parte das aplicações de sistemas inteligentes da Google e vem sendo amplamente adotada por várias empresas que decidiram investir na área de Inteligência Artificial. O fato é que também existem diversos concorrentes, que valem a pena serem explorados. A ênfase deste trabalho foi sobre o TensorFlow, mas existem *frameworks* como PyTorch, PyBrain, Apache SystemML, Apache Mahout, entre outros.

Com este trabalho, foi exemplificada a implementação de uma Rede Neural de Aprendizado Profundo, com foco em uma rede neural para reconhecimento de peças de roupas. Os resultados obtidos com a implementação foram satisfatórios, atingindo um nível de precisão maior que 95%, sendo que alguns casos obtiveram 100% de precisão. A principal meta foi demonstrar como é feita a implementação e como o framework TensorFlow opera e processa os modelos inteligentes através da API do Keras. Nota-se o poder superior da linguagem Python para implementação de algoritmos de inteligência artificial, uma vez que a linguagem base possui um rico conjunto de bibliotecas nativas que possibilitam o desenvolvimento das aplicações de maneira rápida e eficaz.

Ainda há um longo caminho a percorrer, diversos paradigmas a serem quebrados, barreiras de hardware e software a serem rompidas. Uma das grandes barreiras para I.A. é a capacidade de continuar aprendendo em ambientes de produção. Hoje, para ultrapassar esse limite, as empresas como Google coletam constantemente novos dados todos os dias para manter seus sistemas calibrados diante de mudanças na fonte de dados. No entanto, ainda é um trabalho que exige treino e re-treino, que compromete de certa forma a I.A.. O próximo passo é encontrar uma solução para que seja possível treinar algoritmos depois que eles saem do ambiente controlado de um laboratório de software.

7 Referências

ARAÚJO, Flávio HD et al. **Redes Neurais Convolucionais com Tensorflow: Teoria e Prática**. SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. III Escola Regional de Informática do Piauí. Livro Anais-Artigos e Minicursos, v. 1, p. 382-406, 2017.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey (2015). "**Deep learning**". [Acesso em: 20 setembro 2019] Disponível em: <http://adsabs.harvard.edu/abs/2015Natur.521..436L>

YEGULALP, Serdar. "**What is TensorFlow? The Machine Learning Liberary Explained**". [Acesso em: 18 setembro 2019]. Disponível em: <https://www.infoworld.com/article/3278008/tensorflow/what-is-tensorflow-the-machine-learning-library-explained.html>.

TensorFlow. "**Tensors"**. Disponível em: <https://www.tensorflow.org/>. Acesso em 22 de setembro de 2019.

Keras. "**Keras: the Python Deep Learning Library**". Disponível em: < https://keras.io/>. Acesso em 22 de setembro de 2019.

NIELSEN, Michael. "**How the backpropagation works?**". [Acesso em: 19 setembro 2019]. Disponível em: <http://neuralnetworksanddeeplearning.com/chap2.html>

WALKARN, Ujj. "**A Quick Introduction to Neural Networks**". [Acesso em: 19 novembro 2019] Disponível em: <https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/>