UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
TÓPICOS ESPECIAIS EM INTELIGÊNCIA ARTIFICIAL - ELE606

Docente:
José Alfredo Ferreira Costa

Discente:
Douglas Wilian Lima Silva
Roger José Zacarias da Conceição
Semestre 2023.2 - Turma 01

Redes Neurais - MLP (Fashion Mnist)

Natal - RN
Dezembro de 2023

Resumo

O relatório aborda a implementação de Redes Neurais Multilayer Perceptron (MLP) para classificação de vestuário na base de dados Fashion MNIST. Dividido em duas abordagens, o estudo utilizou a biblioteca TensorFlow e uma implementação "do zero". Utilizando TensorFlow, o processo envolveu carregamento e processamento de dados, construção de uma MLP com duas camadas ocultas, treinamento e testes, resultando em alta acurácia de classificação. Na abordagem "do zero", foram realizadas etapas semelhantes, mas com implementação manual das funções de ativação, inicialização de parâmetros e criação de funções para operações de uma MLP.Ambas as implementações demonstraram eficácia na classificação de vestuário, evidenciando a capacidade das MLPs na análise da base Fashion MNIST.

Sumário

Lista de Figuras

1 Introdução

Redes Neurais Multilayer Perceptron (MLP) representam uma classe fundamental de modelos em aprendizado profundo, caracterizadas por sua arquitetura de camadas interconectadas. Inspiradas no funcionamento do cérebro humano, as MLPs consistem em uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada neurônio em uma camada está conectado a todos os neurônios da camada subsequente, permitindo a aprendizagem de padrões complexos e não lineares nos dados. Devido à sua flexibilidade e capacidade de lidar com tarefas variadas, as redes neurais MLP têm sido amplamente aplicadas em problemas de classificação, regressão e reconhecimento de padrões em diversas áreas, destacando-se como uma ferramenta poderosa no campo do aprendizado de máquina.

A aplicação de redes neurais do tipo Multilayer Perceptron (MLP) na análise da base de dados Fashion MNIST traz uma abordagem promissora no campo da visão computacional. A base de dados Fashion MNIST, composta por imagens de artigos de vestuário em escala de cinza, oferece um cenário ideal para o desenvolvimento e avaliação de modelos de aprendizado profundo. As redes neurais MLP, caracterizadas por múltiplas camadas de neurônios interconectados, demonstram eficácia na extração de padrões complexos presentes nas representações visuais das peças de roupa. Este estudo busca explorar a capacidade desses modelos em classificar corretamente diferentes categorias de vestuário, contribuindo assim para avanços significativos na identificação automática de padrões em conjuntos de dados complexos como o Fashion MNIST.

Com base nisso, o presente estudo tem por objetivo a implementação das redes neurais MLP para a classificação dos vestuários presentes na base de dados já citada. A ideia consiste justamente em verificar a eficácia do modelo através da utilização de algoritmos como àqueles presentes na biblioteca TensorFlow.

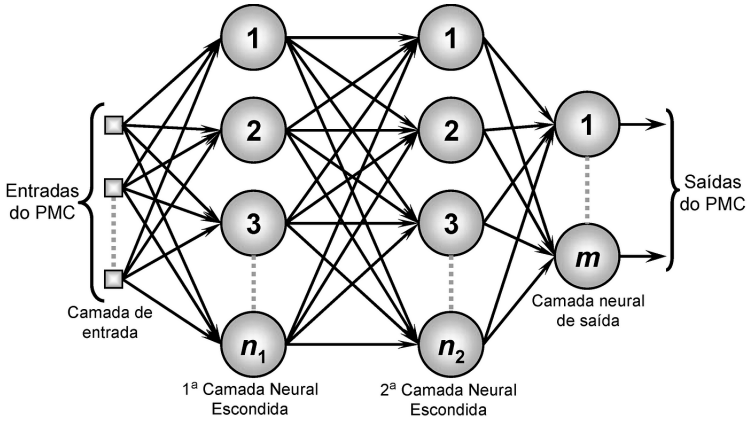

Figura 1: Estrutura MLP. Fonte: MOREIRA.

2 Desenvolvimento

O procedimento de implementação foi realizado em duas partes: utilizando as ferramentas disponíveis na biblioteca TensorFlow e From Scratch utilizando apenas referências e o conhecimento teórico necessário. No entanto, os passos realizados foram basicamente iguais para ambas partes e podem ser identificados abaixo.

- Carregamento de bibliotecas;
- Importação da base de dados;
- Processamento dos dados;
- Adequação para rede neural;
- Implementação da rede;
- Treinamento;
- Plotagem dos resultados.

Através dos passos apresentados foi possível realizar a classificação dos diferentes tipos de vestuários presentes na base Fashion Mnist, disponível tanto na plataforma Kaggle, quanto internamente à biblioteca TensorFlow.

2.1 Carregamento das bibliotecas

Para ambos os casos, diversas bibliotecas foram utilizadas. Não necessariamente as apresentadas aqui foram utilizadas simultaneamente, no entanto, foram essenciais para a construção do algoritmo.

```
1  import pandas as pd
2  import numpy as np
3  import seaborn as sns
4  import matplotlib.pyplot as plt
5  import tensorflow.keras as keras
6  from tensorflow.keras.models import Sequential
7  from tensorflow.keras.layers import Dense
8  from sklearn.model_selection import train_test_split
9  from sklearn.preprocessing import StandardScaler
10 from tensorflow.keras.datasets import fashion_mnist
```

Listing 1: Bibliotecas utilizadas.

A Pandas foi utilizada inicialmente para a criação do dataframe com a base de dados, de forma a permitir uma melhor visualização. A NumPy é essencial para toda a

manipulação matemática necessária tanto para a implementação em conjunto com a TensorFlow, mas principalmente para a implementação from scratch.

As demais bibliotecas utilizadas correspondem para normalização, visualização de gráficos e a própria rede neural em si.

2.2 Visualização da base de dados

A Fashion Mnist consiste em uma base de dados de diferentes categorias de vestuários. Sua estrutura consiste em duas bases: uma de treinamento e uma de teste. Através da Pandas pode-se visualizar sua configuração contendo uma coluna de labels (classes ou categorias) e as demais colunas correspondem a cada valor de pixel de cada imagem presente na base, conforme apresentado na imagem abaixo.

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel7
0	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
2	6	0	0	0	0	0	0	0	5	0	...	0	0	0	30	43	
3	0	0	0	0	1	2	0	0	0	0	...	3	0	0	0	0	
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 785 columns

]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel
0	0	0	0	0	0	0	0	0	9	8	...	103	87	56	0	0	
1	1	0	0	0	0	0	0	0	0	0	...	34	0	0	0	0	
2	2	0	0	0	0	0	0	14	53	99	...	0	0	0	0	63	
3	2	0	0	0	0	0	0	0	0	0	...	137	126	140	0	133	
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 785 columns

Figura 2: Base de dados.

Através dessa estrutura, conhecendo todos os pixels é possível visualizar as imagens através da Matplotlib, podendo verificar como elas estão configuradas e quais as categorias de vestimentas disponíveis na base. Isso traz uma proximidade maior com os dados os quais serão trabalhados, possibilitando uma interpretação mais correta acerca dos resultados obtidos posteriormente.

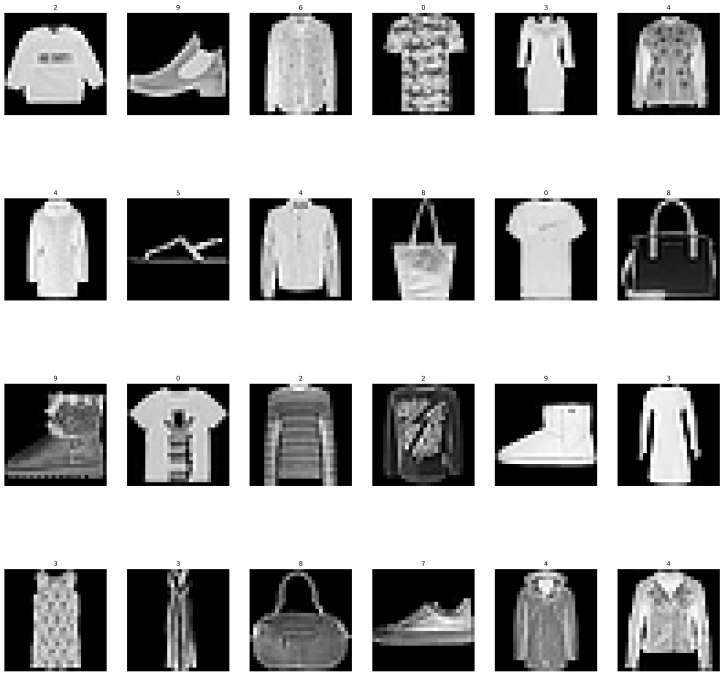

Figura 3: Visualização da base.

2.3 Processamento dos dados

Com os dados devidamente importados e visualizados, a parte de fundamental importância é o seu processamento. Com o processamento feito de forma incorreta, todo o modelo pode ser afetado e desempenhar de forma errônea.

Dessa forma, como processamento dos dados, os mesmos foram separados corretamente entre treinamento e validação e convertidos para matrizes NumPy, obtendo apenas seus valores, sem labels ou nomes de colunas. Além disso, foram verificados os valores máximo e mínimo dos pixels de forma a permitir a normalização dos dados.

Por fim, foi realizada a categorização das classes, transformando cada uma delas em arrays para que pudessem ser introduzidas na rede MLP. Os listings abaixo apresentam o desenvolvimento aqui explicado.

```
#Separacao dos dados de treino e validacao

xtreino = df_treino.drop('label', axis=1).values #conversao para
    matrizes numpy
ytreino = df_treino['label'].values
xval = df_teste.drop('label', axis = 1).values
```

```
yval = df_teste['label'].values
```

Listing 2: Processamento dos dados - 1.

```
#Valores de maximo e minino = 255 e 0

print(xtreino.max())
print(xtreino.min())

xtreino = xtreino / 255
xval = xval / 255

#Categorizacao das classes

num_classes = 10
ytreino = keras.utils.to_categorical(ytreino, num_classes)
yval = keras.utils.to_categorical(yval, num_classes)
```

Listing 3: Processamento dos dados - 2.

2.4 Construção Usando TensorFlow

A utilização da biblioteca TensorFlow agiliza de forma sensacional a implementação de redes neurais multilayer perceptron, ela através da keras, possuem métodos que apenas necessitam ser modificados para cada aplicação.

Em vista disso, o modelo de rede neural foi iniciado usando a keras, em que apenas foram definidas as camadas escondidas (hidden_layers), suas respectivas quantidades de neurônio e a função de ativação utilizada para cada uma delas.

```
model = Sequential()
model.add(Dense(units=512, activation='relu', input_shape=(784,)))
model.add(Dense(units=512, activation='relu'))
model.add(Dense(units=num_classes, activation='softmax'))
model.summary()
```

Listing 4: Inicialização do modelo.

Observa-se que foram utilizados 512 neurônios tanto para a camada inicial, quanto para a interna, e nelas utilizando a função relu como ativação. Também foi definido na camada inicial a forma dos dados de entrada, que nesse caso, possuem 784 linhas, assim como a base de dados. A estrutura da função relu pode ser visualizada na figura 4 abaixo.

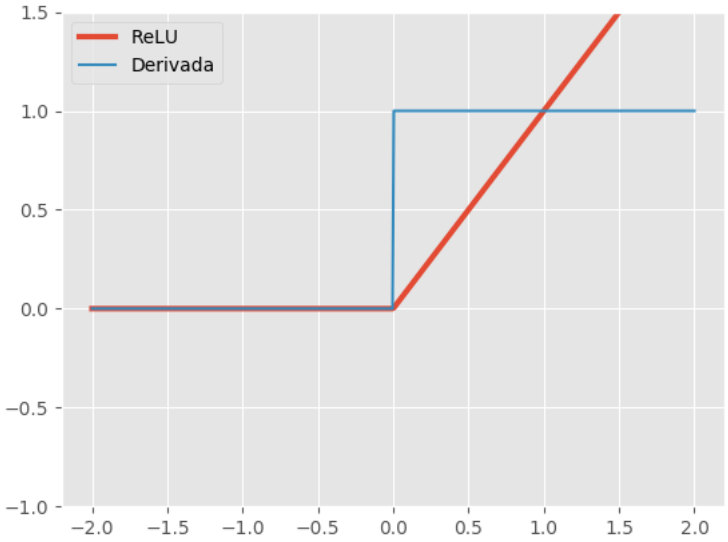

Figura 4: Função ReLu

Por sua vez, para a camada de saída foi utilizada a função softmax, agora com o número de neurônios igual a das classes desejadas, ou seja, um neurônio para cada classe de vestimenta. A função softmax é amplamente utilizada para a classificação porque como cita ("Redes neurais - Funções de ativação - Laboratório iMobilis", 2016) "ela força a saída de uma rede neural a representar a probabilidade dos dados serem de uma das classes definidas. Sem ela as saídas dos neurônios são simplesmente valores numéricos onde o maior indica a classe vencedora". Abaixo, encontra-se a estrutura da função softmax.

$$\phi_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

A função model.summary() apresenta apenas a estrutura de rede criada, conforme a figura abaixo.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               401920

 dense_1 (Dense)             (None, 512)               262656

 dense_2 (Dense)             (None, 10)                5130

=================================================================
Total params: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figura 5: Detalhes do modelo.

Por fim, o modelo foi treinado utilizando o otimizador adam, com uma duração de 20 épocas. Finalmente a acurácia de treino e validação foram respectivamente 93,96% e 90,10%. Os resultados também foram plotados em um gráfico utilizando a matplotlib para uma melhor visualização.

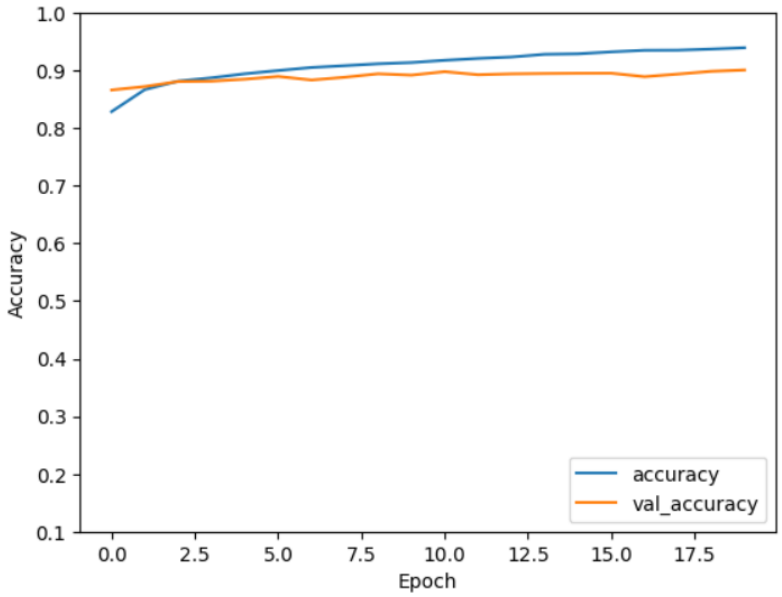

Figura 6: Acurácias do modelo.

2.5 Resultados - TensorFlow

Com o resultado do treinamento apresentado, os dados de teste foram deixados de forma aleatória através da np.random.permutation() e o modelo foi testado com os novos dados visando a verificação da capacidade real do modelo. Abaixo, é possível observar o listing contendo a implementação e a matriz de confusão. Durante esse teste, o modelo obteve uma acurácia de 90,72%.

Vale ressaltar que como o modelo utiliza a softmax como função de saída, é necessário uma pequena alteração para que os resultados possam realmente ser visualizados como classes, dessa forma o listing abaixo também apresenta essa modificação.

```
i = np.random.permutation(df_teste.shape[0])
d = int(0.4*len(i))
test = i[:d]
cjtest = df_teste.loc[test,:]
y = (cjtest['label'].values)
x = (cjtest.drop('label', axis = 1).values)/255
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
ypred = model.predict(x)
```

Listing 5: Randomização do teste.

```
predicted_classes = np.argmax(ypred, axis=1)
accuracy = accuracy_score(y, predicted_classes)
```

Listing 6: Alteração dos parâmetros.

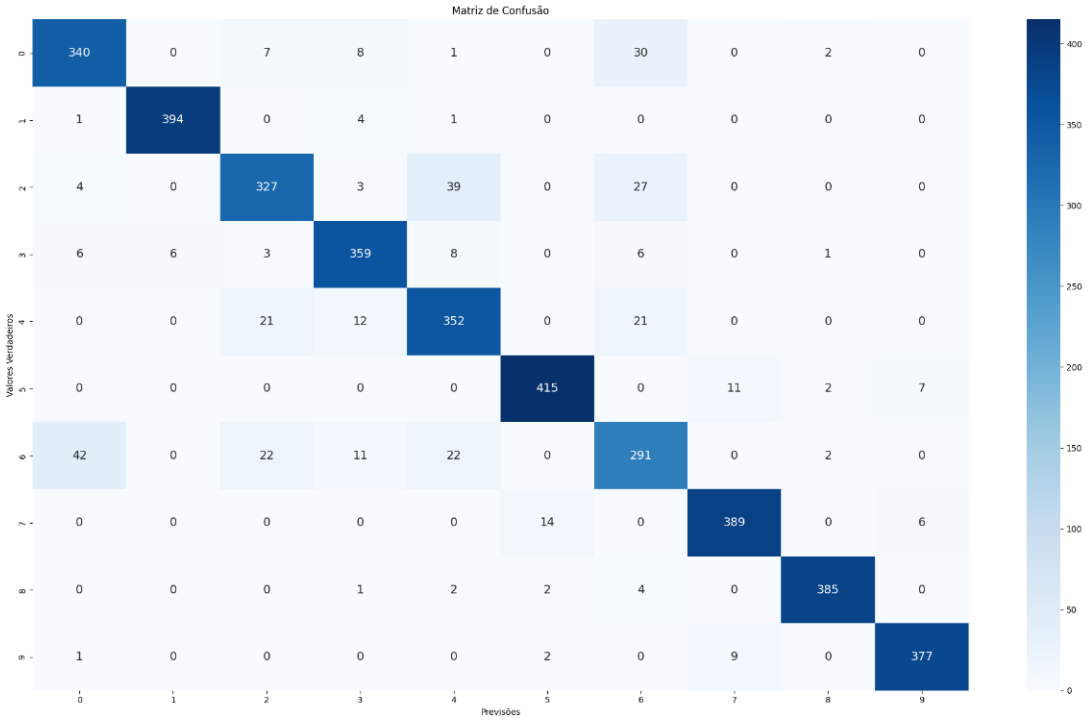

Figura 7: Matriz de confusão.

2.6 Construção From Scratch

Para a implementação da MLP from scratch, ou seja, "do zero", é necessário desenvolver os mesmos passos de processamento de dados tomados para o caso anterior. No entanto, para a categorização das classes, não foi utilizada a biblioteca TensorFlow, mas sim a numpy através da função one.eye(). Outra leve modificação, foi a utilização da biblioteca Scikit-Learn para a normalização dos dados conforme apresentado abaixo.

```
#Utilizacao da Fashion Mnist disponivel na TensorFlow
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

#Redimensionamento das imagens
x_train = x_train.reshape((x_train.shape[0], -1)) / 255.0
x_test = x_test.reshape((x_test.shape[0], -1)) / 255.0

#Normalizacao dos dados para um melhor desempenho
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

```

```
#Categorizando as classes utilizando a NumPy, semelhante ao que foi realizado no codigo anterior com a funcao keras.utils.to_categorical
num_classes = 10
y_train_one_hot = np.eye(num_classes)[y_train]
y_test_one_hot = np.eye(num_classes)[y_test]
```

Listing 7: Processamento.

Posteriormente é necessária a construção da rede neural. Essa parte é bem mais extensa e não será mostrada integralmente aqui, mas pode ser encontrada no repositório do GitHub presente nas referências.

O primeiro passo para se construir uma rede neural é a definição das funções de ativação. Para isso, foram definidas as funções relu e softmax de forma a se equiparar com o caso anterior.

```
# Funcao ReLU
def relu(x):
    return np.maximum(0, x)

# Funcao Softmax
def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)
```

Listing 8: Funções de ativação.

Em seguida, é necessário definir uma série de funções que configurarão a rede proposta. A primeira delas consiste na inicialização dos parâmetros (pesos e bias) que pode ser realizada de diferentes formas. Aqui, foi utilizada a inicialização conhecida como Xavier/Grolot que inicializa os parâmetros como uma gaussiana de média 0 e variância dependente das conexões de saída e de entrada. Para isso, foi utilizada a biblioteca NumPy.

```
def initialize_parameters(input_size, hidden_size, output_size):
    np.random.seed(42)
    W1 = np.random.randn(input_size, hidden_size) * np.sqrt(2/input_size)
    b1 = np.zeros((1, hidden_size))
    W2 = np.random.randn(hidden_size, hidden_size) * np.sqrt(2/hidden_size)
    b2 = np.zeros((1, hidden_size))
    W3 = np.random.randn(hidden_size, output_size) * np.sqrt(2/hidden_size)
    b3 = np.zeros((1, output_size))
    return {'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2, 'W3': W3, 'b3': b3}
```

Listing 9: Inicialização Xavier/Grolot.

As demais funções consistem na foward propagation, que seria a propagação para frente que é responsável por computar a saída da rede neural; a responsável pelo backpropagation que utiliza dos gradientes dos parâmetros e faz a atualização dos pesos; a função que computa as perdas, ou os erros, da rede; e por fim a função de treinamento que se utiliza das demais para realizar o treinamento e a validação da rede MLP.

```
1    def train_mlp(X_train, Y_train, X_val, Y_val, hidden_size=512,
     num_epochs=50, learning_rate=0.001):
2    input_size = X_train.shape[1]
3    output_size = Y_train.shape[1]
4
5    parameters = initialize_parameters(input_size, hidden_size,
     output_size)
6
7    train_losses = []
8    val_losses = []
9    train_accuracies = []
10   val_accuracies = []
11
12   for epoch in range(num_epochs):
13       # Forward pass no conjunto de treinamento
14       train_cache = forward_propagation(X_train, parameters)
15       train_loss = compute_loss(Y_train, train_cache['A3'])
16
17       # Forward pass no conjunto de validacao
18       val_cache = forward_propagation(X_val, parameters)
19       val_loss = compute_loss(Y_val, val_cache['A3'])
20
21       # Calcular acuracia
22       train_accuracy = np.mean(np.argmax(train_cache['A3'], axis=1) ==
      np.argmax(Y_train, axis=1))
23       val_accuracy = np.mean(np.argmax(val_cache['A3'], axis=1) == np.
     argmax(Y_val, axis=1))
24
25       # Armazenar metricas
26       train_losses.append(train_loss)
27       val_losses.append(val_loss)
28       train_accuracies.append(train_accuracy)
29       val_accuracies.append(val_accuracy)
30
31       # Imprimir metricas
32       print(f"Epoch {epoch + 1}/{num_epochs} - Train Loss: {train_loss
     :.4f} - Val Loss: {val_loss:.4f} - Train Acc: {train_accuracy:.4f} -
     Val Acc: {val_accuracy:.4f}")
33
34       # Backward pass e atualizacao dos parametros
35       backward_propagation(X_train, Y_train, parameters, train_cache,
```

```
    learning_rate)
```

Listing 10: Função de treino.

2.7 Resultados - From Scratch

Com a implementação mostrada acima, foi realizado o treinamento da rede e observado que sua acurácia não estava satisfatória com as 20 épocas inicialmente propostas. Dessa forma, foram definidas 50 épocas e realizado o treinamento novamente de forma a obter uma acurácia de treinamento de 42,73% e de validação de 42,04%. A figura 8 apresenta os resultados.

Isso mostra que a função criada manualmente necessitaria de mais épocas para atingir a acurácia de uma implementação já bem desenvolvida como a do TensorFlow, no entanto, isso ocuparia muito mais tempo já que seria necessário rodar por algo em torno de 100 épocas e também mais custo computacional. No entanto, o desempenho obtido ainda pode ser considerado satisfatório.

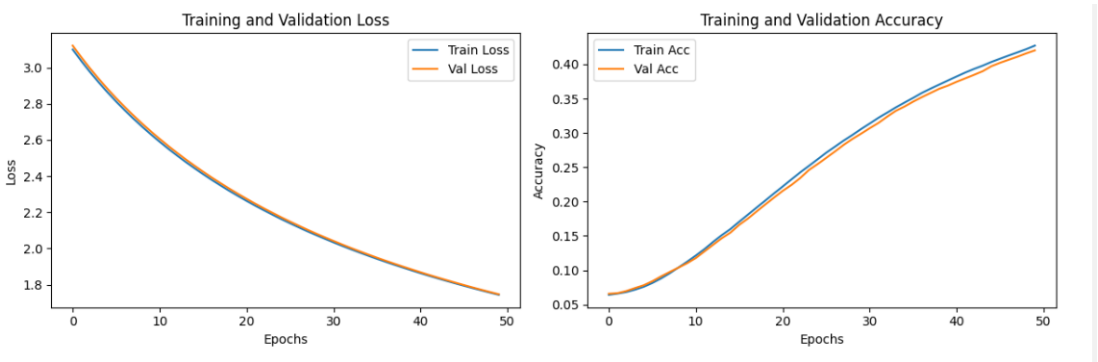

Figura 8: Acurácias e perdas.

3 Considerações Finais

O estudo apresentou resultados promissores na aplicação de Redes Neurais Multilayer Perceptron (MLP) para a classificação de vestuário na base de dados Fashion MNIST. A implementação utilizando TensorFlow demonstrou a eficácia e conveniência dessa biblioteca para a construção, treinamento e teste de modelos de aprendizado profundo.

A abordagem "do zero"também revelou a viabilidade de construir uma MLP manualmente, evidenciando a compreensão detalhada dos componentes e operações fundamentais de uma rede neural.

Ambas as implementações alcançaram bons níveis de acurácia na classificação de vestuário, mas claro com um desempenho bem superior para a implementação usando TensorFlow, destacando a capacidade das MLPs em lidar com problemas complexos de reconhecimento de padrões em imagens.

É importante ressaltar que, embora a abordagem utilizando TensorFlow seja mais prática e rápida, a implementação manual permitiu um entendimento mais profundo das operações subjacentes a uma MLP, proporcionando intendimento valiosos sobre o funcionamento interno dessas redes.

Em síntese, este estudo reforça a importância das Redes Neurais Multilayer Perceptron como uma poderosa ferramenta no campo da visão computacional e reconhecimento de padrões, oferecendo alternativas de implementação que se adaptam a diferentes necessidades e níveis de compreensão técnica.

Referências

[1] MOREIRA, S. Rede Neural Perceptron Multicamadas. Disponível em: Link de acesso.

[2] SHMILOVICH, K. Building a multilayer perceptron from scratch. Disponível em: Link de acesso. Acesso em: 14 dez. 2023.

[3] UNIVERSIDADE FEDERAL DE SANTA CATARINA CENTRO TECNOLÓGICO DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA Guilherme Góes Mendonça. [s.l: s.n.]. Disponível em: Link de acesso. Acesso em: 14 dez. 2023.

[4] Redes neurais - Funções de ativação - Laboratório iMobilis. Disponível em: Link de acesso. Acesso em: 14 dez. 2023.

[5] Funções de Ativação. Disponível em: Link de acesso. .

[6] SILVA, Douglas Wilian Lima. Tópicos em IA. GitHub. Disponível em: Link de acesso
.