Comparação de modelos de redes neurais artificiais CNN e MLP, para um caso de teste de reconhecimento ótico de caracteres

RESUMO

Algumas topologias e parâmetros diferentes para uma rede neural artificial convolucional (CNN) são comparados com um modelo do tipo Perceptron Multi-Camadas (MLP), utilizando o dataset MNIST para reconhecimento ótico de caracteres. Redes neurais têm sido usadas para resolver uma grande variedade de tarefas que são difíceis de resolver utilizando programação baseada em regras comuns, incluindo visão computacional e reconhecimento de voz [2]. Este trabalho utilizou a biblioteca TensorFlow para comparar alguns modelos CNN e dentre eles selecionar os dois com melhor acurácia, e também foi comparado com o modelo MLP.

PALAVRAS-CHAVE

Aprendizado de máquina, Redes neurais artificiais, Inteligência artificial, Computação científica

1 INTRODUÇÃO

Redes neurais artificias são algoritmos que tentam de alguma forma se assemelhar a neurônios biológicos, e usam alguns conceitos como sinapses e transmissão de sinais processados para outros neurônios conectados. Os sinais são representados na forma de números, e a saída de um neurônio é o resultado de um cálculo feito por uma função não linear que leva em consideração a soma das entradas do neurônio. Tipicamente neurônios possuem pesos e *bias* que precisam ser ajustados em uma etapa anterior conhecida como treinamento. Os pesos aumentam ou diminuem a intensidade do sinal na conexão. Neurônios podem ser configurados para possuírem um limiar aonde o sinal é enviado somente se ultrapassar aquele limiar.

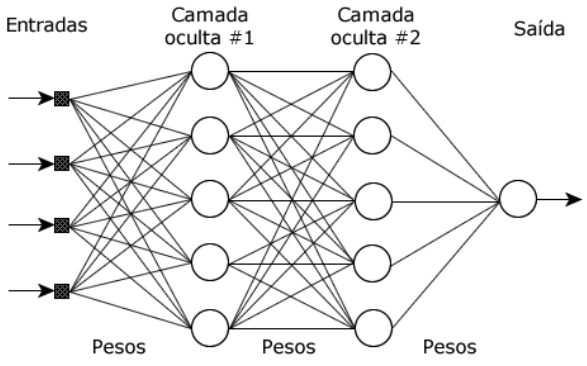

Figura 1: Rede MLP.

A Figura 1 mostra uma rede neural artificial usando a topologia MLP (Perceptron Multi-Camadas) que possui uma camada de entrada, duas camadas ocultas, e uma de saída com um neurônio. A Figura 2 mostra um exemplo usando

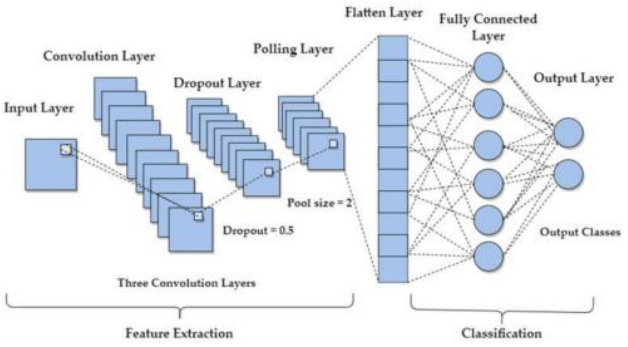

Figura 2: Rede CNN.

a topologia CNN (rede neural convolucional) que inclui camadas de extração de recursos como convolução, *dropout*, *pooling*, e *flatten*.

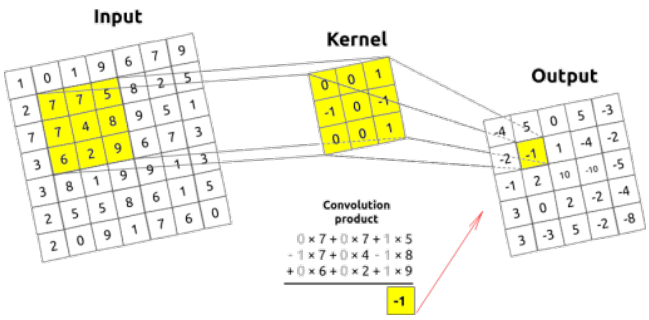

Figura 3: Convolução.

Na rede CNN um exemplo de camada de convolução é mostrado na Figura 3, e consiste em usar um kernel de convolução na camada de entrada para produzir um tensor de saídas.

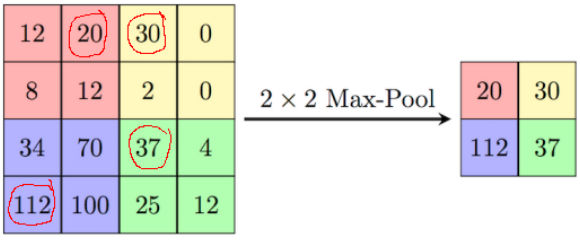

Figura 4: *Pooling*.

Um exemplo de *pooling* é mostrado na Figura 4 e consiste em reduzir a amostra de entrada ao longo de suas dimensões

espaciais (altura e largura) tomando o valor máximo em uma janela de entrada para cada canal de entrada. A janela é deslocada a passos largos ao longo de cada dimensão.

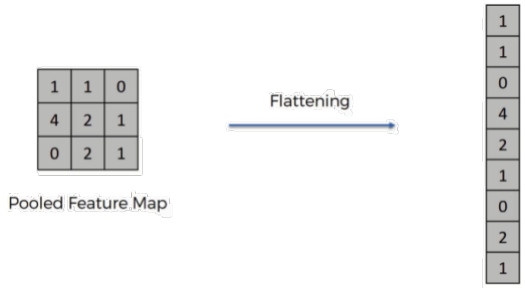

Figura 5: Flattening.

Um exemplo de *fattening* é mostrado na Figura 5 e consiste em mudar o formato dos dados e deixá-los achatados.

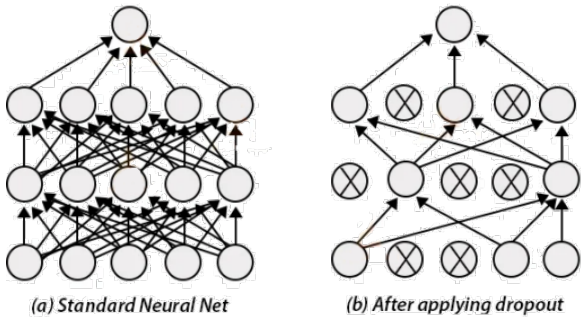

Figura 6: Dropout.

Um exemplo de *dropout* é mostrado na Figura 6 e consiste em desativar alguns neurônios durante o treinamento, para tentar reduzir o *overfitting*.

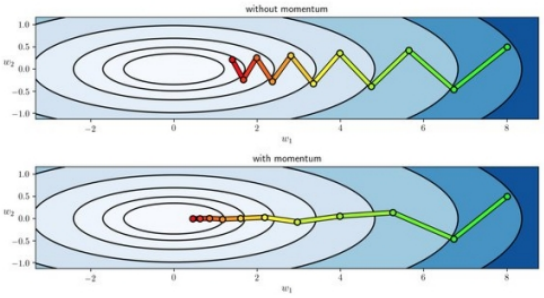

Figura 7: Momentum.

Alguns parâmetros que podem ser usados durante o treinamento são o Momentum (Figura 7) e a regularização (Figura 8). O momentum é utilizado quando se usa o algoritmo de otimização de gradiente descendente, onde é possível que a superfície que a equação descreve seja complexa e com vários mínimos locais, dificultando a obtenção do mínimo global. Neste caso pode-se usar o termo de momentum para aumentar o tamanho dos passos dados em direção ao mínimo, desta forma tentando saltar os mínimos locais.

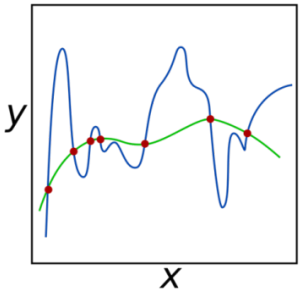

Figura 8: Regularização.

A regularização pode ser utilizada para evitar o *overfiting* dos dados, estabelecendo um limite para os pesos evitando a saturação de sinapses.

2 CLASSIFICAÇÃO DE IMAGENS

O dataset MNIST (*Modified National Institute of Standards and Technology*) consiste em um grande conjunto de dígitos escritos à mão digitalizados e classificados, é usualmente utilizado para treinamento de sistemas de processamento de imagens, e também em aprendizado de máquina. Possui 60 mil imagens para treinamento e 10 mil para teste, e cada imagem tem 20x28 pixels usando níveis de cinza. A Figura 9 mostra uma representação de uma imagem e a correspondente rede neural artificial.

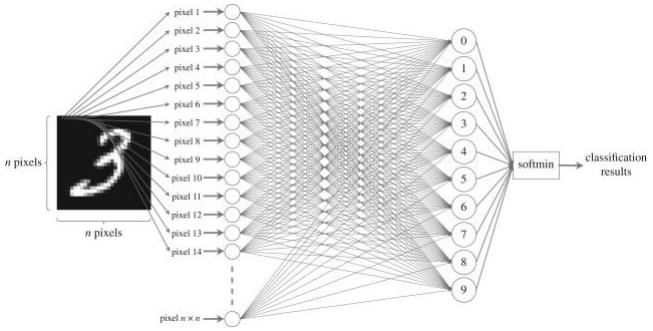

Figura 9: Dataset MNIST e rede neural artificial. Fonte:royalsocietypublishing.org.

3 IMPLEMENTAÇÃO

Para a rede CNN, a implementação da primeira topologia mostrada na Listagem 1 foi feita usando a biblioteca TensorFlow e foi definida como mostrado na Figura 10 com uma camada de entrada seguida por uma de convolução, uma de *pooling*, uma de *flattening*, uma *dense* (camada totalmente

conectada e o tipo mais comum de camada usado em modelos perceptron multicamadas), uma *dropout*, e finalmente uma camada de saída).

Listagem 1: Topologia 1 - definição das camadas da rede neural artificial.

```
model = Sequential()
model.add(
    Conv2D(
        32,  # number or filters (output)
        (3, 3),  # kernel size
        activation="relu",
        kernel_initializer="he_uniform",
        input_shape=in_shape))  # (28, 28, 1)
model.add(MaxPool2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation="relu",
          kernel_initializer="he_uniform"))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation="softmax"))
```

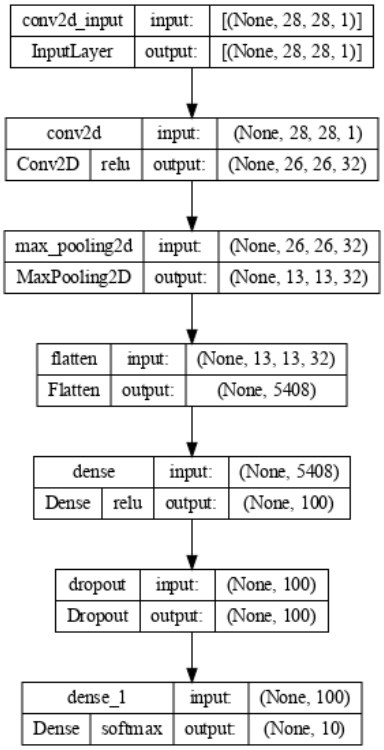

Figura 10: Topologia 1.

A função de ativação usada foi a ReLU (Figura 11) que retorna o próprio valor se ele for negativo, ou então zero se for negativo. Como ela retorna zero para valores negativos, tende a "apagar"alguns neurônios durante o treinamento. Da mesma forma, valores positivos podem "explodir"durante o treinamento pois não impõe um valor limite.

O argumento `kernel_initializer` define os pesos aleatórios iniciais das camadas de neurônios, durante o treinamento. O valor `he_uniform` faz com que sejam utilizadas amostras

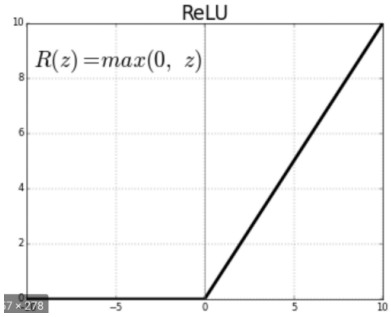

Figura 11: Função de ativação ReLU.

de uma distribuição uniforme onde os limites são dados por: $limit = \sqrt{\frac{6}{\texttt{fan\_in}}}$ onde `fan_in` é a quantidade de unidades de entrada no tensor de peso.

O argumento input_shape é usado para definir a camada de entrada. A função `MaxPool2D` define o *pooling*, `Flatten` achata os dados, `Dense` define uma camada totalmente conectada de neurônios, `Dropout` faz com que alguns neurônios sejam "desativados"para tentar evitar o *overfitting*, e finalmente na camada de saída temos a ativação `softmax` que converte um vetor de valores em uma distribuição probabilística sendo que os elementos do vetor de saída estão no intervalo (0,1) e somam 1.

Listagem 2: Topologia 1 - configuração do modelo de treinamento

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

A definição do modelo de treinamento é mostrado na Listagem 2. O otimizador `adam` é um algoritmo que implementa o método estocástico de gradiente descendente baseado na estimativa adaptativa de momentos de primeira e de segunda ordem. O parâmetro `sparse_categorical_crossentropy` define o cálculo da perda de entropia cruzada entre os rótulos e as previsões, quando há duas ou mais classes de rótulos. A métrica `accuracy` calcula a frequência com que as previsões são iguais aos rótulos.

Listagem 3: Topologia 1 - treinamento do modelo

```
history = model.fit(x_train_norm,
                    y_train,
                    epochs=100,
                    batch_size=128,
                    validation_data=(x_test_norm, y_test),
                    verbose=0)
```

A definição e o treinamento são mostrados na Listagem 3. A quantidade de épocas de treinamento foi definida como 100. O `batch_size` define o número de amostras por atualização de gradiente (128). O `validation_data` são os dados sobre os quais é feita a validação da perda e quaisquer métricas do modelo no final de cada época.

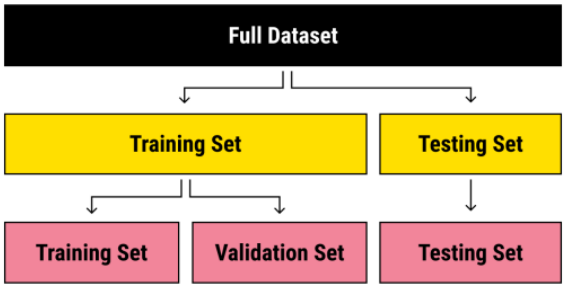

Figura 12: Divisão do dataset em conjuntos de treinamento, validação, e teste.

A Figura 12 mostra a divisão usual do conjunto de dados em Treinamento, Validação, e Teste. Para este trabalho, no entanto, como a intenção era avaliar a validação, somente dois conjuntos foram utilizados, um de Treino e um único de Validação/Teste.

4 ANÁLISE

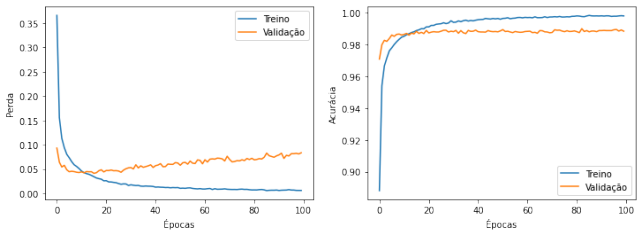

Figura 13: Perda e acurácia para a topologia 1.

A Figura 13 mostra os gráficos de perda e acurácia para a Topologia 1 descrita na seção anterior. A acurácia obtida foi de 0.988. A perda usando dados de validação (teste) tem a tendência de ir aumentando ao longo das épocas, tendência inversa do treinamento. As próximas topologias são uma variação desta primeira.

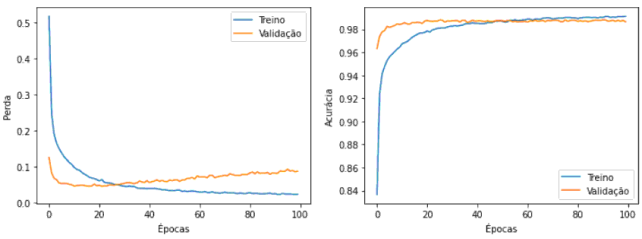

Figura 14: Perda e acurácia para a topologia 2.

A Figura 13 mostra os gráficos de perda e acurácia para a Topologia 2. A diferença entre a topologia 1 e 2 está na camada escondida que foi reduzida de 100 neurônios para 50.

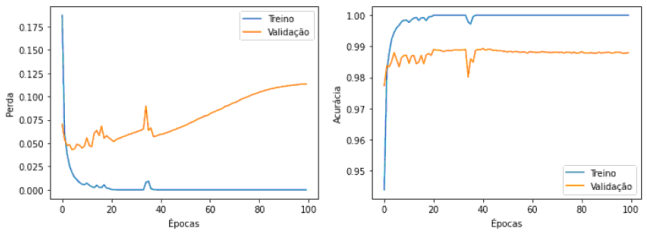

Figura 15: Perda e acurácia para a topologia 3.

A acurácia obtida foi de 0.987, um pouco menor que a topologia anterior. O treino demora um pouco mais para chegar no mínimo de perda, quando comparado com a topologia 1.

A Figura 15 mostra os gráficos de perda e acurácia para a Topologia 3. Nesta topologia a camada Dropout foi eliminada. A acurácia obtida foi de 0.988, similar à primeira topologia, porém a perda durante a validação é grande.

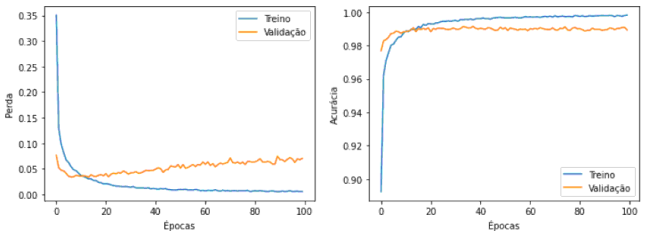

Figura 16: Perda e acurácia para a topologia 4.

A Figura 19 mostra os gráficos de perda e acurácia para a Topologia 4. Nesta topologia o tamanho do kernel foi aumentado de 3x3 para 5x5. A acurácia obtida foi de 0.989, maior do que as topologias anteriores.

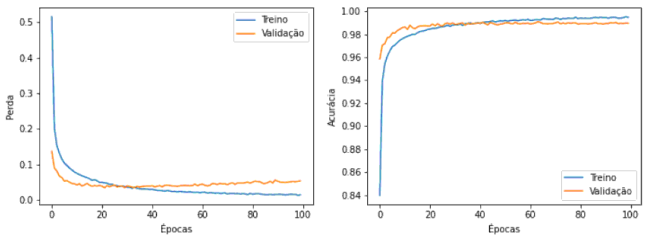

Figura 17: Perda e acurácia para a topologia 5.

A Figura 17 mostra os gráficos de perda e acurácia para a Topologia 5. Nesta topologia o Maxpool (pool) passou de 2x2 para 5x5. Assim como a topologia anterior (4), aumentou a acurácia com relação às primeiras topologias.

A Figura 18 mostra os gráficos de perda e acurácia para a Topologia 6. Nesta topologia a quantidade de filtros de saída na camada convolucional (Conv2D) passou de 32 para 16. A acurácia não aumentou, como aconteceu com a topologia anterior.

A Figura 19 mostra os gráficos de perda e acurácia para a Topologia 7. Nesta topologia foi alterado de 32 para 16

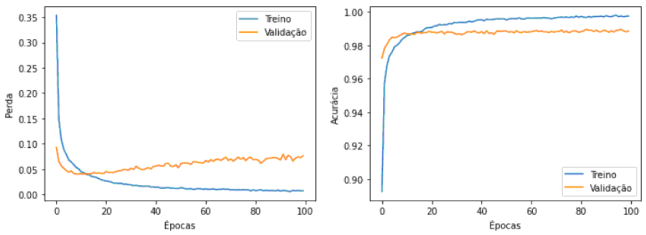

Figura 18: Perda e acurácia para a topologia 6.

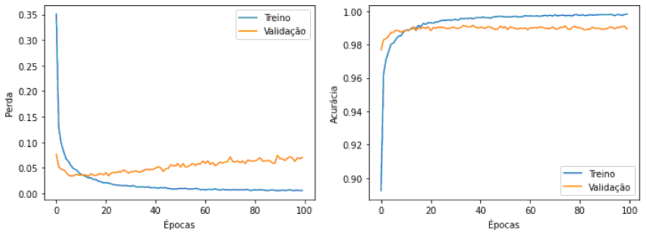

Figura 19: Perda e acurácia para a topologia 4.

Tabela 1: Acurácias obtidas nos experimentos. Os maiores valores estão em vermelho.

Topologia	Acurácia
1	0.988
2	0.987
3	0.988
4	**0.989**
5	**0.989**
6	0.988
7	0.985

filtros em Conv2D, e de 100 para 30 neurônios na camada intermediária (Dense). A acurácia piorou.

A Tabela 1 mostras as acurácias obtidas nos experimentos realizados, onde foram usados topologias e parâmetros diferentes. As duas melhores acurácias foram obtidas nas topologias 4 e 5. Para estas duas, foram feitos os gráficos de matriz de confusão, e um heatmap mostrando um teste de McNemar para as duas topologias.

A Figura 20 mostra a matriz de confusão comparando a predição com o valor real, para a topologia 4. Como a acurácia foi relativamente alta, a linha diagonal concentra a maior quantidade de predições corretas.

A Figura 21 mostra a matriz de confusão comparando a predição com o valor real, para a topologia 5. Da mesma forma que para a topologia 4, como a acurácia foi relativamente alta, a linha diagonal concentra a maior quantidade de predições corretas.

A Figura 22 mostra o teste de McNemar comparando a predição com o valor real, para os dois melhores modelos (4 e 5). Este gráfico também mostra na linha diagonal os valores relativamente altos de predições corretas.

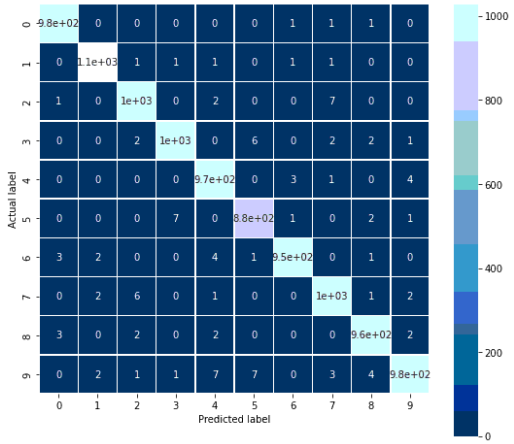

Figura 20: Matriz de confusão para a topologia 4.

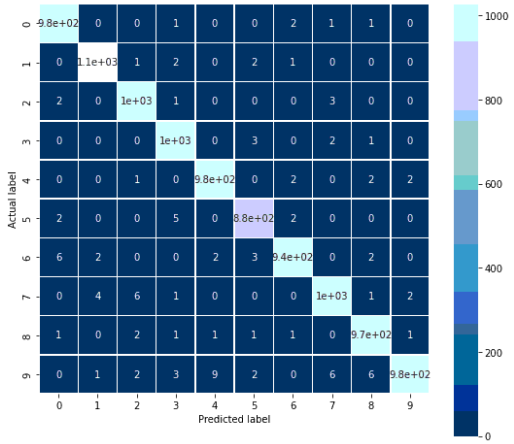

Figura 21: Matriz de confusão para a topologia 5.

A Figura 23 mostra os gráficos de perda e acurácia para a Topologia MLP. Esta topologia mostrou que durante o treinamento a acurácia foi equivalente à rede CNN, porém para os dados de treinamento a acurácia foi pior.

A Figura 24 mostra dois gráficos sobrepostos, MLP e CNN. As curvas de treinamento em azul estão próximas, porém as de validação estão distantes, mostrando uma maior acurácia da topologia CNN.

5 COMENTÁRIOS FINAIS

O trabalho mostrou algumas topologias para a rede CNN e uma topologia MLP, e fez comparações entre elas. O dataset MNIST foi escolhido, e a biblioteca TensorFlow foi utilizada. A comparação CNN x MLP mostrou uma maior acurácia para a rede CNN utilando os dados de validação. Nas topologias CNN não houve grande variação na acurácia, porém

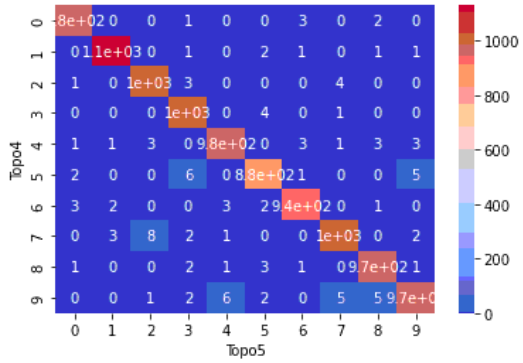

Figura 22: Teste de McNemar para os dois melhores modelos (4 e 5).

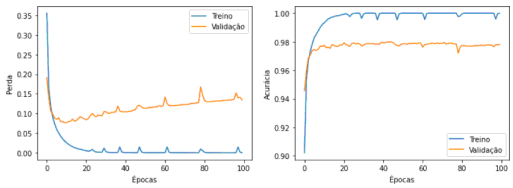

Figura 23: Resultado para a topologia MLP.

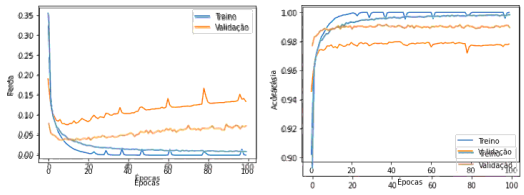

Figura 24: Sobreposição dos gráficos MLP e CNN.

as duas topologias com maior acurácia foram selecionadas e comparadas.

REFERÊNCIAS

[1] [n.d.]. Monthly Car Sales in Quebec 1960. https://www.kaggle.com/datasets/dinirimameev/monthly-car-sales-in-quebec-1960.

[2] [n.d.]. TensorFlow - plataforma completa de código aberto para aprendizagem de máquina. https://www.tensorflow.org/.

[3] Jason Brownlee. [n.d.]. Seasonal Persistence Forecasting With Python. https://machinelearningmastery.com/seasonal-persistence-forecasting-python.

[4] Jason Brownlee. [n.d.]. TensorFlow 2 Tutorial: Get Started in Deep Learning with tf.keras. https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/.

[5] Jason Brownlee. [n.d.]. Time Series Forecasting with the Long Short-Term Memory Network in Python. https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/.

[6] Bengio Goodfellow and Courville. 2016. Deep learning. https://www.deeplearningbook.org.

[7] I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep Learning*. MIT Press. https://books.google.com.br/books?id=Np9SDQAAQBAJ

[8] S.S. Haykin. 2009. *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall. https://books.google.com.br/books?id=K7P36lKzI_QC

[9] Rob Hyndman. [n.d.]. tsdl: Time Series Data Library. https://pkg.yangzhuoranyang.com/tsdl/index.html.

[10] Barış Karaman. [n.d.]. Forecasting the monthly sales with LSTM. https://towardsdatascience.com/predicting-sales-611cb5a252de.

[11] Jonasz Karwacki. [n.d.]. Boston Housing. https://www.kaggle.com/code/escofresco/boston-housing.

[12] Avatar Senad Kurtiši. [n.d.]. Univariate Time Series Forecasting of Car Sales. https://github.com/senadkurtisi/Univariate-Time-Series-Forecasting.

[13] Keenan M. [n.d.]. Predicting House Prices using a Deep Neural Network: Case with the Boston Dataset. https://www.thepythonacademy.com/post/predicting-house-prices-using-a-deep-neural-network-case-with-the-boston-dataset.

[14] Prof. M. G. Quiles. [n.d.]. Notas de aula CAP 351.

[15] Dawid Stodolkiewicz. [n.d.]. Tensorflow 2 – Regression on the Boston Housing Dataset. Part 1 – Basics. http://www.stodolkiewicz.com/2020/01/04/tensorflow-2-regression-on-boston-housing-dataset/.