Exercícios de Fixação de Conceitos 3 - EFC3 - IA048

Marcelo Eduardo Pederiva RA: 122580

Parte 1 - Classificação Binária com redes MLP

a)

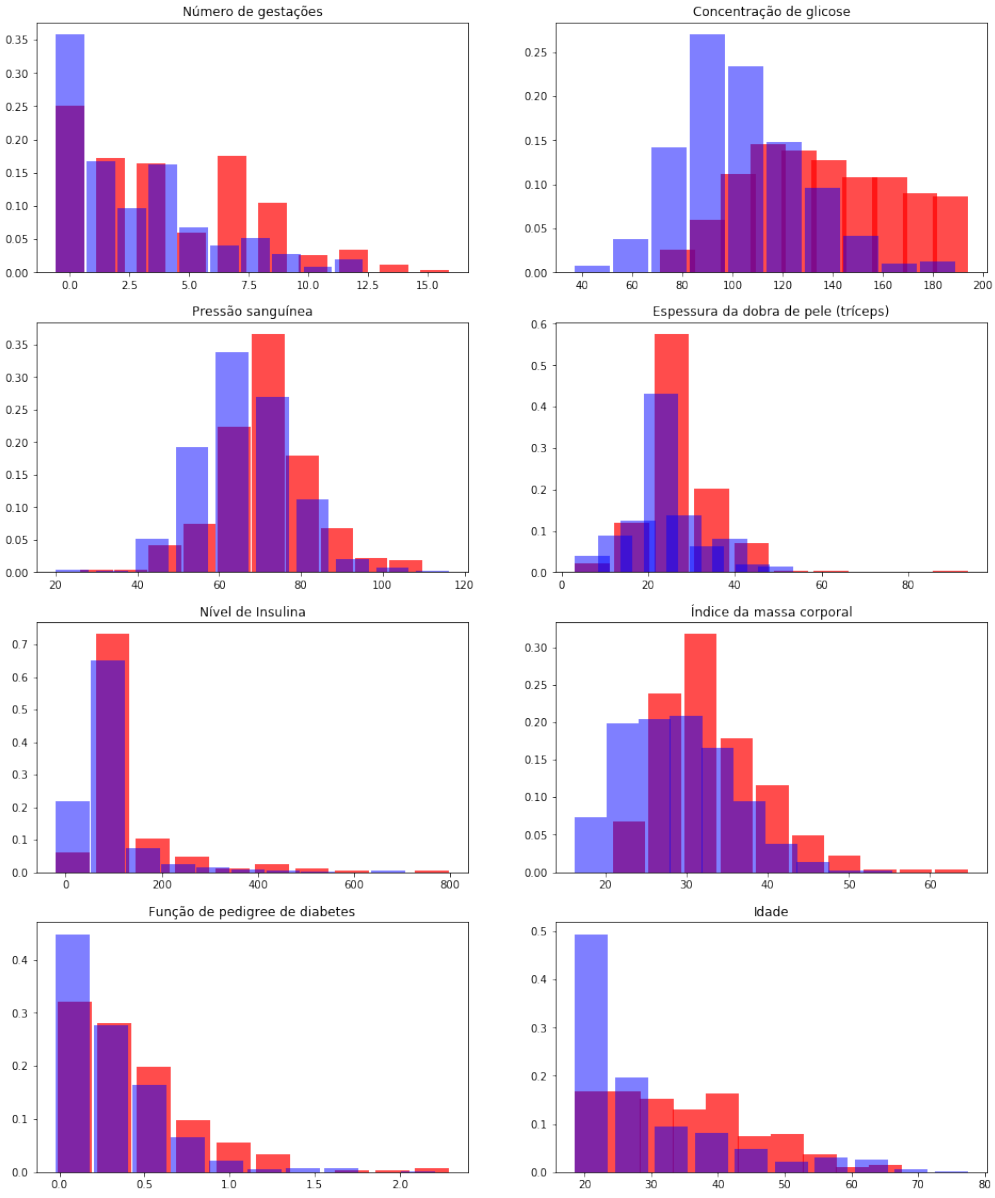


Figura 1: Histogramas, as barras vermelhas representam diagnostico positivo para diabetes e as barras azuis negativo.

Inicialmente, para a análise dos dados, foi observado um desbalanceamento entre a quantidade de pessoas com diagnostico positivo para diabetes e as diagnosticadas negativamente. No qual, a quantidade de dados de pessoas diagnosticadas sem diabetes representam quase o dobro da quantidade de dados classificados de forma contrária. Dessa forma, para observar o a relevância de cada atributo para a classificação de cada classe, foi feito uma normalização do total de cada classe para cada atributo. Em outras palavras, o histograma apresentado pela Figura 1 demonstra, em seu eixo y, porcentagem de pessoas para cada atributo.

Como podemos observar pelos dados, o problema apresenta um grande desafio, uma vez que os atributos não apresentam grandes diferenças para representação de um tipo de diagnostico. Podemos observar que a concentração de glicose representa um dos principais fatores de distinção entre as classes. Já o histograma do Nivel de Insulina demonstra um resultado próximo dentre as duas classificações.

Entretanto, os histogramas se diferenciam em alguns casos, como Idade e Número de Gestações. Nestes atributos, o histograma de uma categoria apresenta um pico em uma região, enquanto a outra possui uma distribuição não acentuada. Esses casos provavelmente irão ter grande relevância para a classificação correta do diagnostico apos o treinamento da MLP.

b)

Para o treinamento da Máquina, inicialmente foi feito uma normalização dos dados de pela técnica Min-Max:

$$X' = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{1}$$

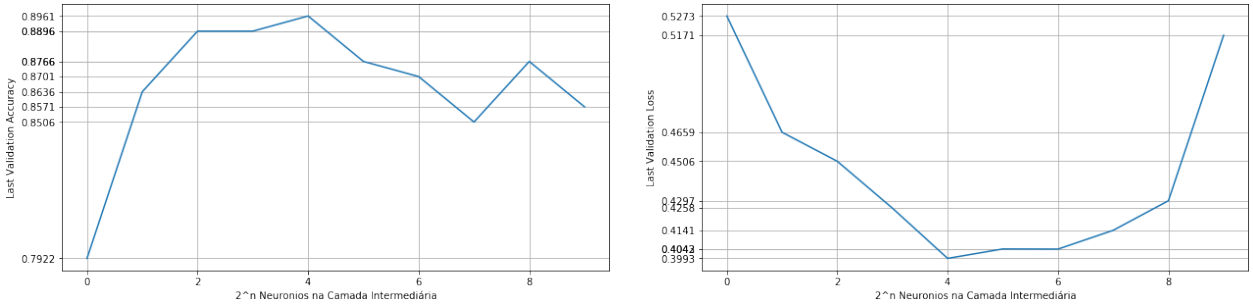
A seguir, para o treinamento foi separado os dados entre dados de treinamento e dados de validação (*Holdout*). Para isso, levando em conta o desbalanceamento das classes, foi separado 20% dos dados diagnosticados positivamente mais 20% dos dados contrários, para validação. Da mesma forma, foi somado os 80% de cada classe como base de dados de treinamento.

Após o embaralhamento dos dados, foi feito o treinamento de uma MLP com 1 camada intermediária. Nessa etapa, foi variado os números de neurônios na camada intermediária dentre os seguintes valores:

$$16 — 32 — 64 — 128 — 256 — 512 — 1024 — 2048 — 4096 — 8192$$

O treinamento de todos modelos foi feito com o cálculo do *loss* = *"sparse_categorical_crossentropy"*, o otimizador = "adam", *bach_size* = 64, *epochs* = 200 e um "*callback*"de *EarlyStopping* monitorando o mínimo do "*loss_validation*". Dessa forma, o *callback* buscou interromper o treinamento para que o modelo apresentasse o mínimo de *overfitting*.

Após o treinamento encontramos uma variação de acurácia e do *loss* dos dados de validação de acordo com cada modelo.



(a) Acurácia frente aos dados de Validação de acordo com cada modelo.

(b) Loss frente aos dados de Validação de acordo com cada modelo.

Figura 2: Comparação dos modelos MLP.

Como podemos observar na Figura 2, o modelo que utilizou 2^4 (256) neurônios na camada intermediária apresentou o melhor resultado, tanto nos valores de acurácia, quanto nos dados de *loss*.

c)

Com o número de neurônios definidos no item b) utilizamos o modelo com 256 neurônios para treinar em 400 épocas (*epochs*). Com os outros parâmetros definidos da mesma forma, neste momento o *callback* utilizado foi o de *ModelCheckpoint*, o qual permite um salvamento dos pesos treinados cada vez que o valor de *loss_validation* diminuísse (ou o valor de *acuraccy_validation* aumentasse), tendo assim os pesos com o epoch que permitiu o mínimo de *overfitting*, ou máximo de acurácia.

Na Figura 3 temos a evolução do treino deste modelo.

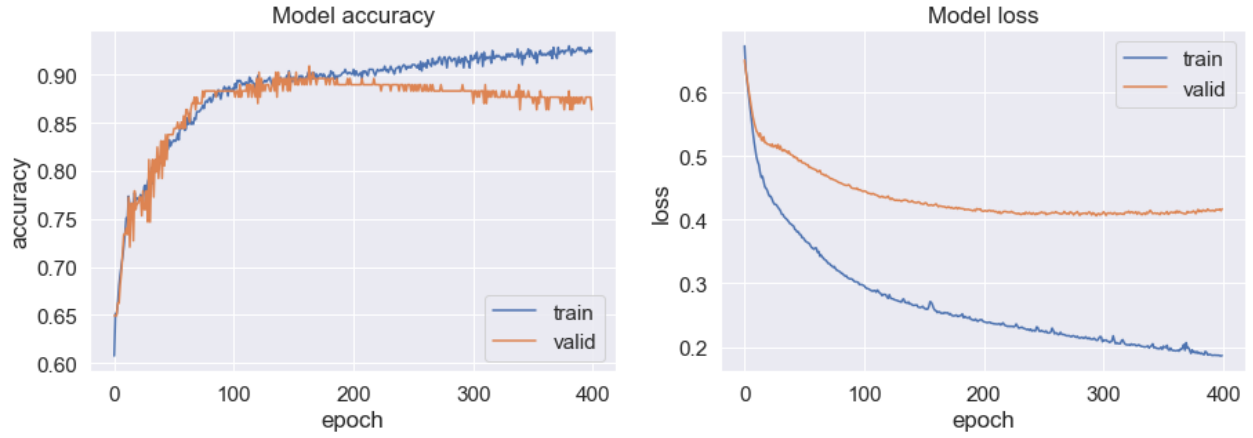


Figura 3: Evolução do treinamento do MLP.

Podemos observar que a acurácia de validação se mantém constante após, aproximadamente, a época 150, enquanto a acurácia de treinamento continua subindo. Este mesmo fenômeno pode ser observado com as curvas de *loss*, com o *loss* de treinamento

diminuindo. Esse contraste entre os dois valores de acurácia demonstra o máximo que o treinamento alcançou antes de apresentar *overfitting*.

Com o *callback* (ModelCheckpoint) salvando os pesos no menor *loss_validation* e maior *accuracy_validation*, garantimos o salvamento dos pesos que represente o melhor modelo em acurácia ou *loss* frente aos dados de validação.

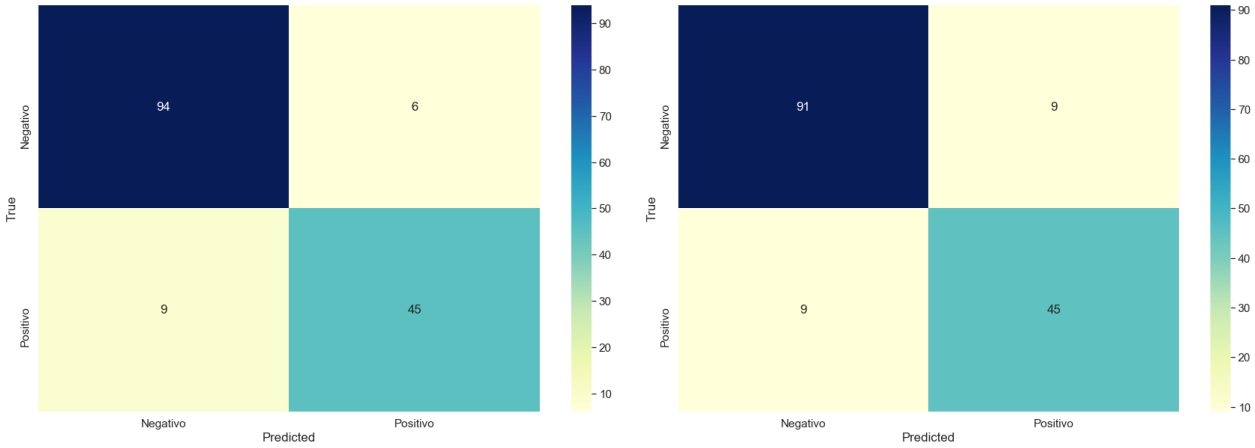
Assim, para a análise dos modelos em diferentes aspectos. O treinamento salvou o peso que apresentasse o menor *loss* e a *maior* acurácia com os dados de validação.

O modelo apresentou o seguinte resultado:

Minimum Loss Checkpoint:	Best Accuracy: 88.31 %	Loss : 0.406362
Maximum Accuracy Checkpoint:	Best Accuracy: 90.26 %	Loss : 0.426401

É interessante observar como os resultados divergem entre a melhor acurácia e o melhor *loss*, demonstrando que para obter um modelo com o menor *overfitting* não será, necessariamente, o modelo com a maior acurácia.

Por fim, temos a Matriz de Confusão destes dois modelos. Ambos apresentam resultados bem próximos, com um maior número de acerto dos casos negativos devido aos desbalanceamento dos dados.



(a) Maximum Accuracy Checkpoint.

(b) Minimum Loss Checkpoint.

Figura 4: MLPs treinadas.

Parte 2 - MNIST, MLP e CNN

a)

Para a análise das MLPs com quantidades diferentes de camadas, foi definido os seguintes modelos:

Modelo 1:

tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dense(10, activation='softmax')

Modelo 2:

tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(10, activation='softmax')

Modelo 3:

tf.keras.layers.Flatten(),
tf.keras.layers.Dense(1024, activation=tf.nn.relu),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(256, activation=tf.nn.relu),
tf.keras.layers.Dense(10, activation='softmax')

Modelo 4:

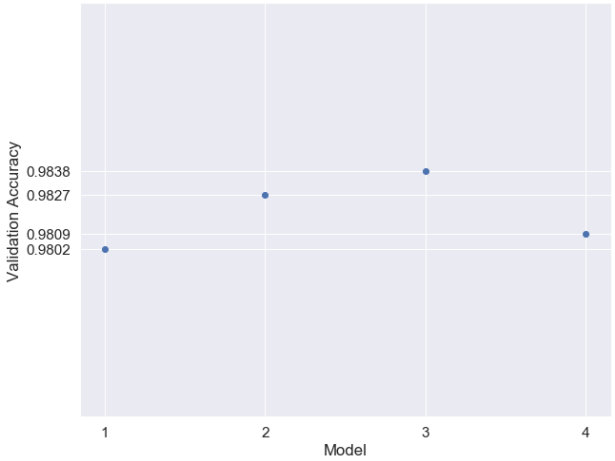
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(1024, activation=tf.nn.relu),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Dense(256, activation=tf.nn.relu),
tf.keras.layers.Dense(10, activation='softmax')

Modelo 1,2 e 3 com 1,2,3 camadas intermediárias, respectivamente. Para observação do *BatchNormalization*, foi implementado um 4° modelo onde foi adicionado 1 camada a mais, de *BatchNormalization*, no meio das camadas anteriores, apenas para observar se este modelo irá apresentar um desempenho melhor que a encontrada no modelo 3.

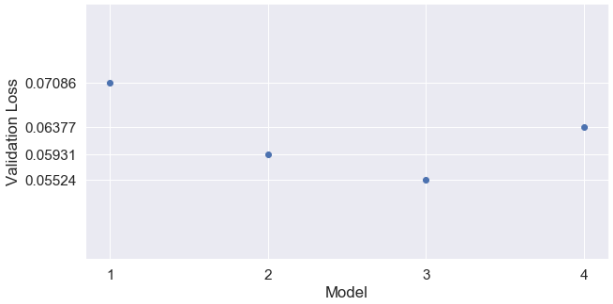
Todos os modelos foram treinados com os mesmos parâmetros:

— Otimizador = "adam"
— Loss = "sparse_categorical_crossentropy"
— epochs=10
— batch_size = 64
— callback = EarlyStopping, interrompendo o treinamento no mínimo *loss_validation*

Com todos modelos treinados, podemos observar o desempenho de cada um deles de acordo com suas máximas acurácias e mínimos valores de *loss*, frente aos dados de teste.



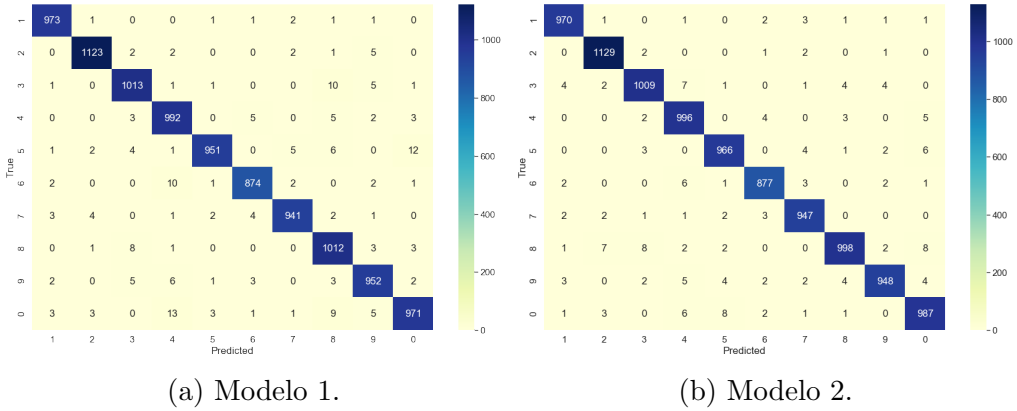
(a) Acurácia de cada modelo.



(b) Loss de cada modelo.

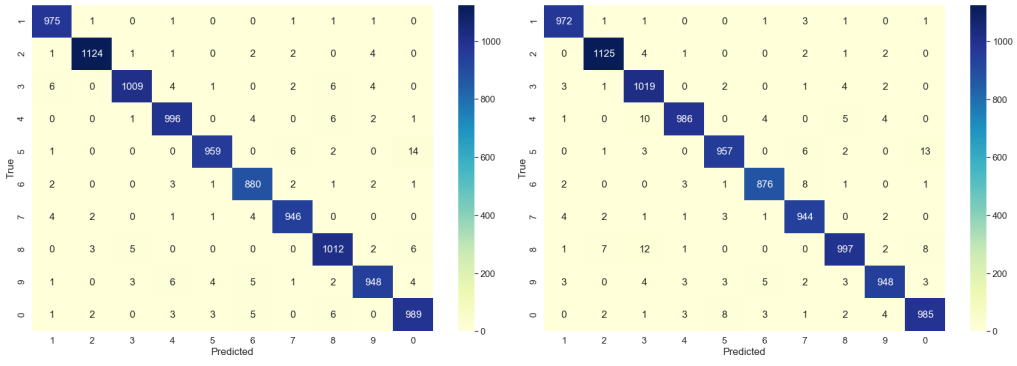
Figura 5: Comparação dos modelos.

Observamos que a implementação do Dropout resultou num aumento de performance do modelo 1 para o 2 e o aumento de neuronios junto com a implementação de outra camada com 256 neurônios melhorou ainda mais esta performance. O 4° modelo, com a implementação do BatchNormalization apresentou um decaimento na performance do modelo 3. Sendo assim, o Modelo 3 se destacou dentre os outros pela maior acurácia e menor valor de *loss*.



(a) Modelo 1.

(b) Modelo 2.



(c) Modelo 3.

(d) Modelo 4.

Figura 6: Comparação dos modelos.

Observamos pelas Matrizes de Confusão, que o Modelo 3 possui um desempenho muito próximo das demais. Apesar de não apresentar o maior número de acertos em todas classes, o modelo manteve uma maior precisão geral comparada com os outros.

b)

1)

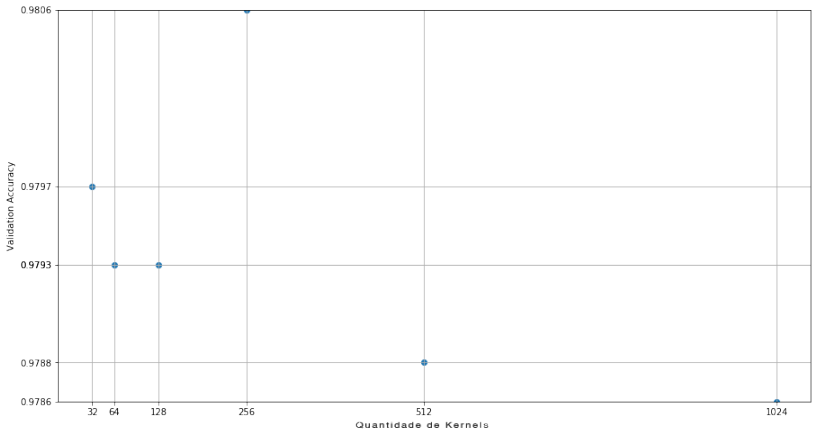
Nesta etapa, iremos implementar uma CNN. O modelo possui a seguinte arquitetura:

Conv2D(number, kernel_size=(size, size), activation='relu', input_shape=(28, 28, 1))
MaxPooling2D(pool_size=(2, 2))
Flatten()
Dense(10, activation='softmax')

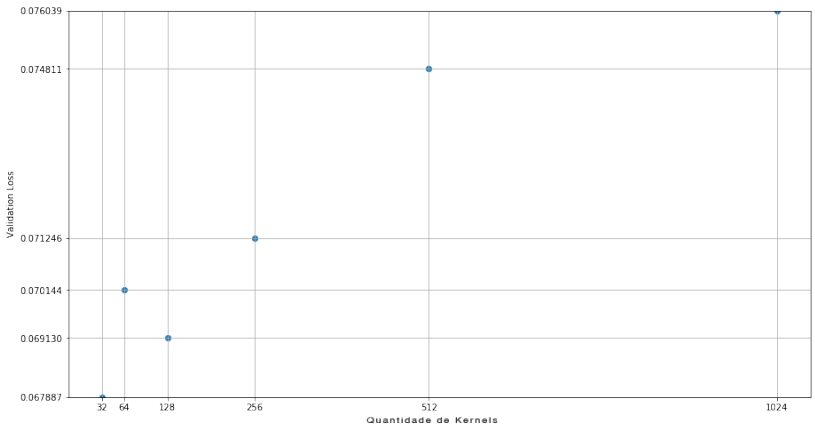
Neste item, utilizaremos um tamanho fixo de *kernel*, $size = 2$, e iremos variar a quantidade de *kernels* (*number*) presentes na camada convolucional. As quantidades serão variadas dentro dos seguintes valores:

$$32 \text{ — } 64 \text{ — } 128 \text{ — } 256 \text{ — } 512 \text{ — } 1024$$

Para o treinamento foi utilizado os mesmos parâmetros do item a).



(a) Acurácia de cada modelo.



(b) Loss de cada modelo.

Figura 7: Comparação das quantidades de *kernels*.

Como podemos observar na Figura 7, os modelos que apresentaram melhores acurácias não apresentaram os melhores valores de *loss*. Entretanto, os valores de *loss* com até 256 *kernels* tiveram uma baixa variação. Além disso, observando os valores de acurácia, o modelo que usou 256 *kernels* se mostrou o mais preciso.

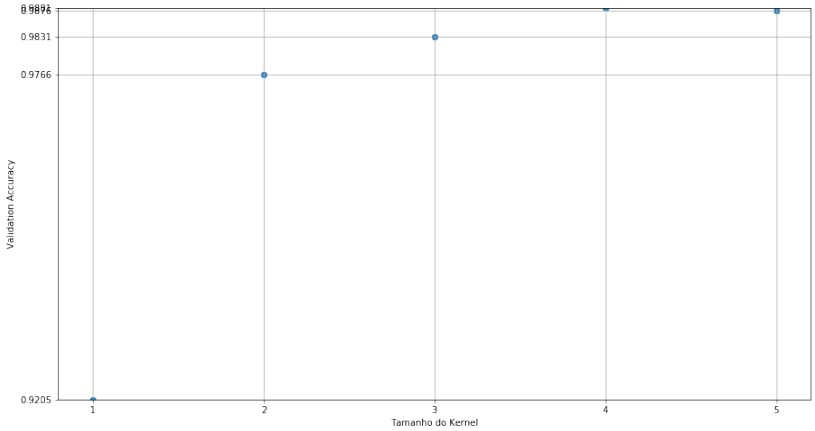
Levando em conta os dois fatores, foi escolhido o modelo com 256 *kernels*, pela relação geral entre a acurácia e o valor de *loss*.

2)

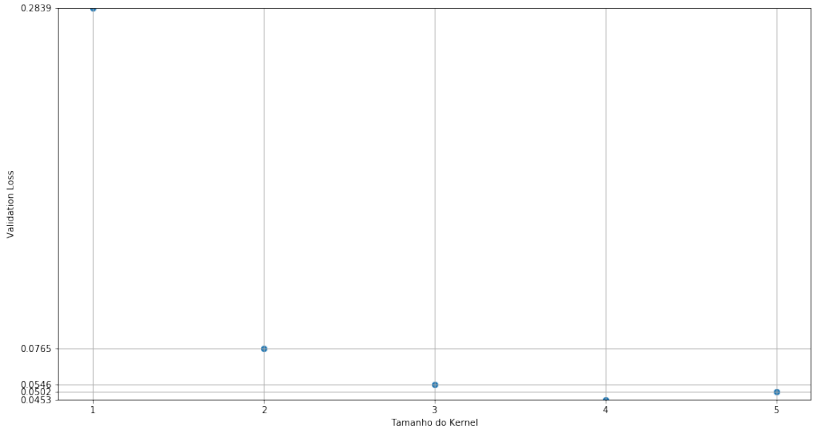
Tendo definido a quantidade de *kernels*, fixamos este parâmetro ($number = 256$) e variamos o tamanho dos *kernels*($size$) entre os valores:

$$1 — 2 — 3 — 4 — 5$$

Usando os mesmos parâmetros de treinamento do item anterior, obtemos o seguinte resultado.



(a) Acurácia de cada modelo.



(b) Loss de cada modelo.

Figura 8: Comparação dos tamanhos do *kernel*.

O desempenho do aumento do tamanho do *kernel* apresentou uma curva, onde o máximo de acurácia e mínimo de erro se manifestou no mesmo modelo, quando foi utilizado o $size = 4$.

Sendo assim, este parâmetro foi escolhido para seguir com a atividade.

c)

Utilizando os resultados obtidos pelos itens anteriores, montamos a seguinte CNN:

Conv2D(256, kernel_size=(4, 4), activation='relu', input_shape=(28, 28, 1))
MaxPooling2D(pool_size=(2, 2))
Flatten()
Dense(10, activation='softmax')

Com o treinamento monitorando o mínimo *loss_validation*, o modelo apresentou o seguinte desempenho:

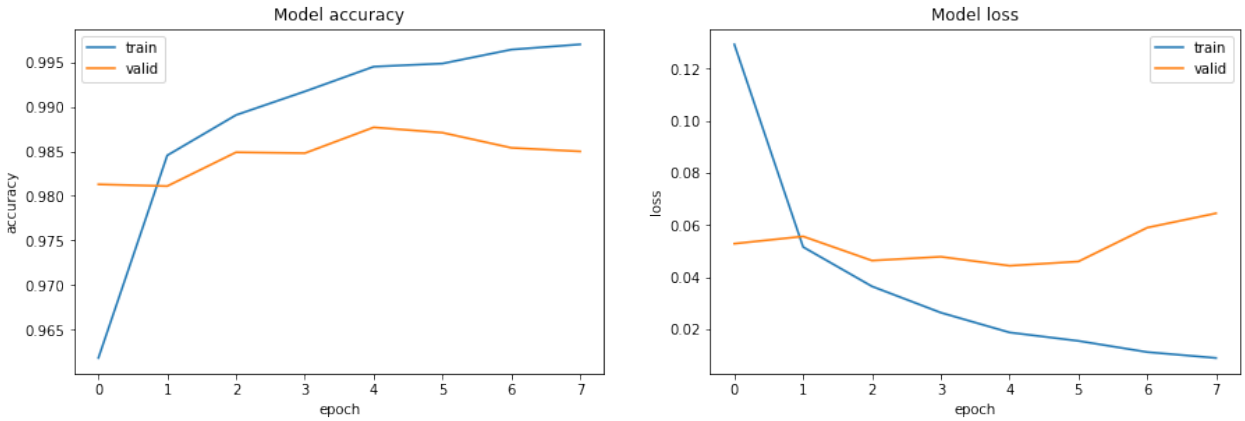


Figura 9: Evolução do treinamento da CNN.

Acurácia, usando os dados de validação = 98.66% Loss = 0.0531

Acurácia Global = 98.50%

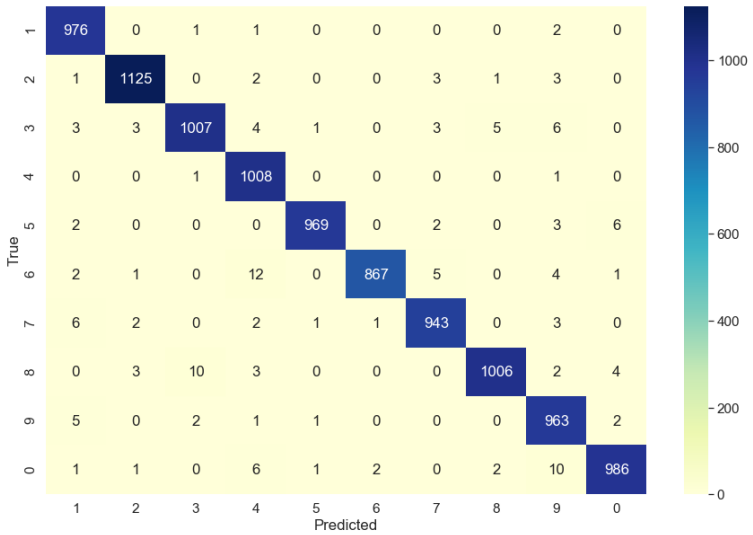
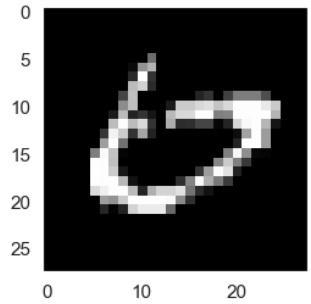
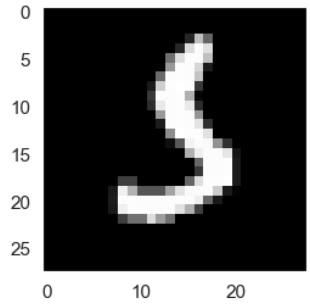


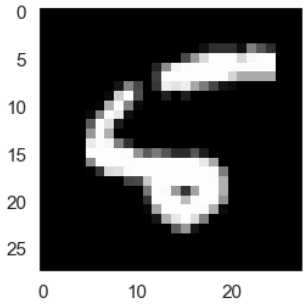
Figura 10: Matriz de Confusão da CNN, para os dados de teste.



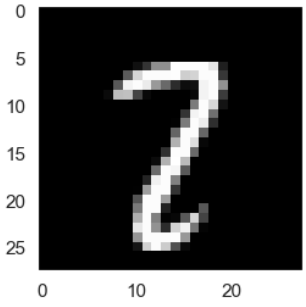
(a) Predito : 0, Real : 6.



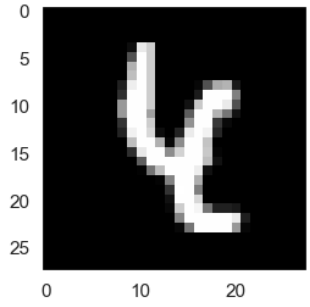
(b) Predito : 3, Real : 5.



(c) Predito : 6, Real : 5.



(d) Predito : 7, Real : 2.



(e) Predito : 8, Real : 4.

Figura 11: Classificações feitas incorretamente.

O treinamento desta CNN simples resultou numa ótima performance. A implementação de camadas convolucionais apresentou uma melhora no desempenho comparado às MLPs treinadas no item a).

Como podemos observar na Figura 10 e 11, apesar do ótimo desempenho, a CNN ainda teve dificuldades em categorizar corretramente alguns números. Os valores apresentados pela Figura 11a,11c e 11d poderiam confundir até um humano, no momento de identificar o número. Entretanto, a Figura 11e demonstra uma grande diferença de característica entre o padrão do desenho do número 8 e o 4. Isto implica quem, provavelmente, alguma região da imagem induziu a rede a categorizá-la de forma incorreta.

Supondo superficialmente, poderíamos acreditar que o encontro das retas no centro do desenho do número 8 foi classificado pela rede como uma característica principal do número. Levando ela a acreditar que a Figura 11e representasse o padrão do número 8.

d)

Nesta etapa foi definido uma arquitetura maior para a CNN, com até 3 camadas convolucionais. Assim, utilizando os resultados dos itens anteriores, foi definido o seguinte modelo:

Conv2D(256, kernel_size=(4, 4), activation='relu', input_shape=(28, 28, 1))
Conv2D(512, kernel_size=(2, 2), activation='relu')
Dropout(0.5)
MaxPooling2D(pool_size=(2, 2))
Conv2D(128, kernel_size=(1, 1), activation='relu')
Flatten()
Dense(10, activation='softmax')

Novamente, monitorando o mínimo do valor de *loss_validation*, o modelo foi treinado.

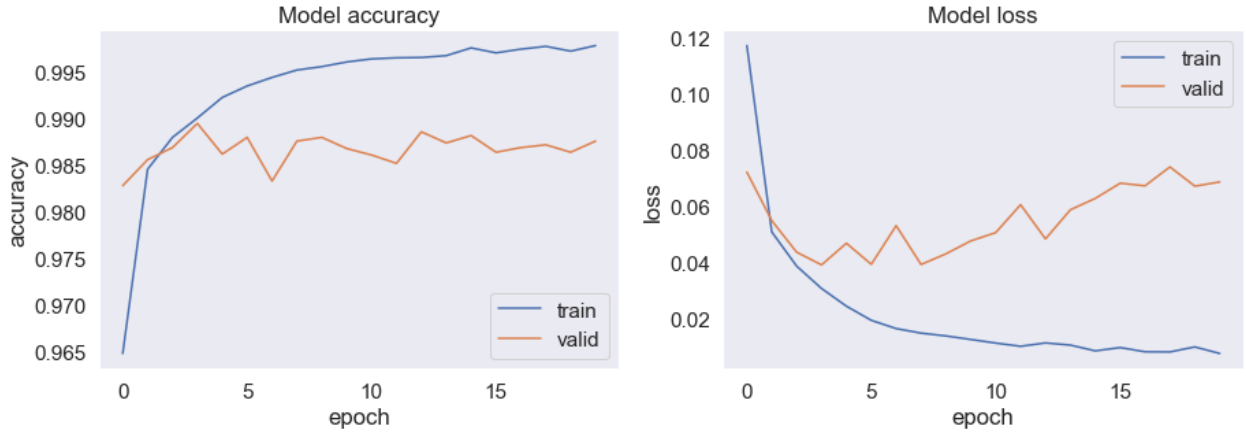


Figura 12: Evolução do treinamento da CNN com 3 camadas convolucionais.

Acurácia, usando os dados de validação = 98.95% Loss = 0.0388

Acurácia Global = 98.95%

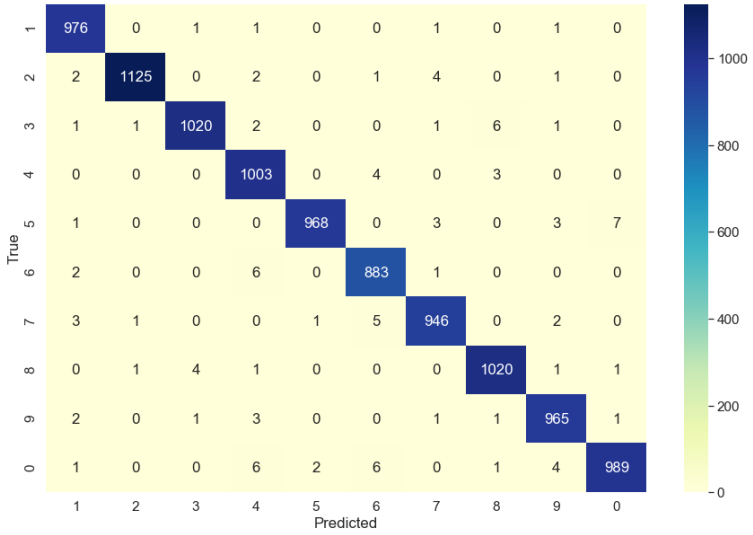
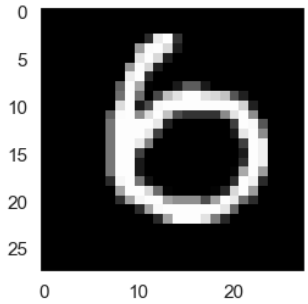
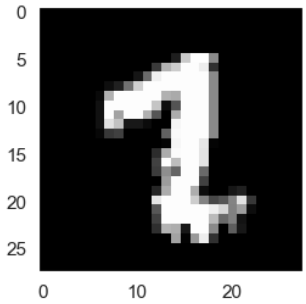


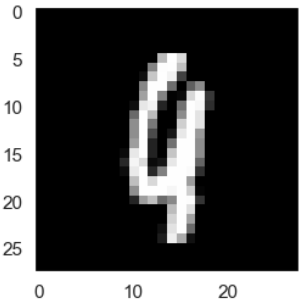
Figura 13: Matriz de Confusão da CNN com 3 camadas convolucionais, para os dados de teste.



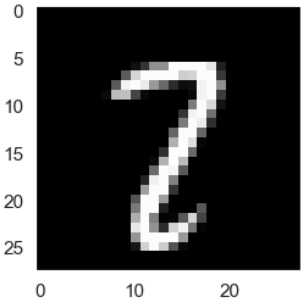
(a) Predito : 0, Real : 6.



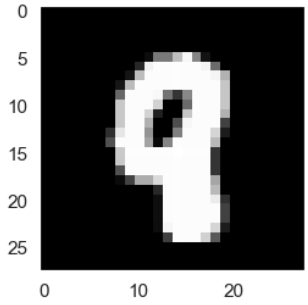
(b) Predito : 1, Real : 2.



(c) Predito : 4, Real : 9.



(d) Predito : 7, Real : 2.



(e) Predito : 9, Real : 8.

Figura 14: Classificações feitas incorretamente.

O treinamento desta CNN apresentou uma melhora na performance de acurácia e *loss*, frente aos dados de teste, quando comparamos com o modelo do item c). Apesar da melhora, a rede ainda apresenta algumas classificações erradas como demonstrado pela Figura 13 e 14. Da mesma forma que no item anterior, algumas dessas classificações erradas são compreensíveis, já que alguns desenhos confundem até os humanos no momento de classificação.

Por fim, o trabalho mostrou o avanço no desempenho de classificação conforme a variação dos modelos de redes neurais (Tabela 1).

Tabela 1: Comparação dos desempenhos.

Modelos	Acurácia de Teste (%)	*Loss* de Teste	Acurácia Global
MLP	98.38	0.0552	98.38
CNN Simples	98.66	0.0531	98.50
CNN 3 camadas convolucionais	98.95	0.0388	98.95

Apesar da ultima arquitetura ter alcançado quase 99% de precisão nos dados de teste, atualmente existem inúmeros modelos, com diversas camadas, que buscam sempre melhorar esse desempenho.