Patrick de Carvalho Tavares Rezende Ferreira - 175480 EA072 - EFC1

Questão 1

Coeficiente de regularização utilizado (taxa de acertos):

675.58805031572205734846647829

Coeficiente de regularização para erro quadrático: 55.71523605095193687475330080

A matriz confusão obtida pelo classificador na questão 1 está expressa abaixo:

2119,	12,	3,	3,	7,	0,	9,	25,	1,	0
94,	1628,	40,	33,	6,	76,	44,	56,	5,	33
41,	61,	1739,	6,	37,	25,	35,	46,	44,	16
32,	5,	3,	1744,	14,	18,	5,	18,	91,	3
27,	14,	160,	34,	1313,	65,	23,	79,	54,	48
33,	19,	1,	15,	12,	1821,	0,	14,	1,	27
76,	11,	20,	41,	3,	0,	1862,	3,	96,	28
178,	25,	75,	28,	64,	22,	12,	1507,	57,	23
19,	7,	39,	118,	5,	1,	132,	12,	1581,	19
2,			11,	19,	22,	4,	27,	1,	1902

Tabela 1: Matriz de confusão do classificador linear.

Cada linha da matriz representa, respectivamente, de 1 a 10, cada classe pretendida pelo treinamento (onde a classe 10 representa o dígito zero) e, nas colunas, quantas vezes cada classe, também de 1 a 10, foi escolhida pelo classificador quando este deveria escolher aquela da referida linha.

Nota-se que a classe reconhecida com maior precisão pelo sistema após o treinamento foi o dígito 1, sendo que a classe que mais gerou equívocos foi a do dígito 5, muitas vezes confundida com o dígito 3.

Nas figuras 1 e 2, temos exemplos de dígitos que foram reconhecidos erroneamente pelo classificador. Na figura 1, percebe-se que a curva superior distorcida e a curva inferior exageradamente grande fizeram o dígito 3 ficar parecido com o 5, fazendo o classificador se confundir. Para a figura 2, a parte superior do 5 foi feita de maneira muito discreta, fazendo o classificador classificá-lo como um 3.

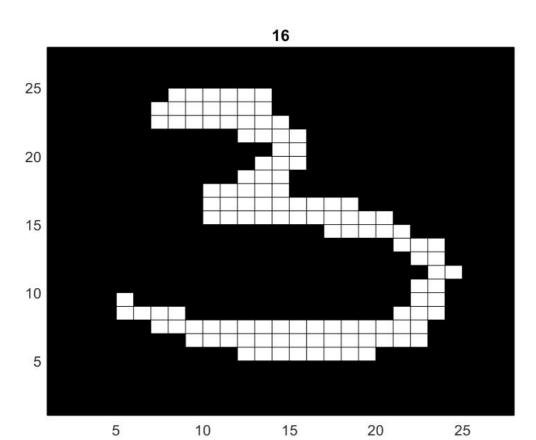


Figura 1: Um dígito três que foi reconhecido como cinco.

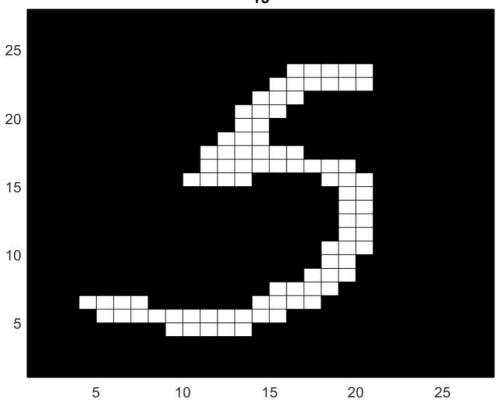


Figura 2: Um dígito cinco que foi reconhecido como três.

Gráficos de busca do coeficiente de regularização no anexo 1.

Gráficos de calor da matriz W no anexo 2.

Questão 2

Coeficiente de regularização para erro quadrático: 24.25146506416636427161392930

Q2.1)

A matriz confusão obtida pela ELM na questão 2 está expressa abaixo:

2140,	11,	3,	3,	7,	6,	1,	5,	2,	1
18,	1824,	28,	21,	6,	27,	29,	32,	7,	23
11,	52,	1819,	2,	44,	17,	22,	49,	25,	9
20,	7,	0,	1780,	5,	14,	5,	8,	94,	0
19,	11,	76,	11,	1575,	49,	12,	14,	25,	25
4,	14,	0,	14,	17,	1860,	1,	12,	2,	19
34,	26,	6,	31,	6,	1,	1962,	3,	54,	17
45,	24,	58,	11,	47,	17,	17,	1714,	37,	21
9,	8,	26,	63,	12,	4,	63,	24,	1712,	12
1,	11,	3,	7,	11,	16,	1,	19,	3,	1927

Tabela 2: Matriz de confusão da ELM.

A matriz de confusão da ELM segue a mesma distribuição da questão 1. Nota-se que a taxa de acertos em todas as classes aumentou, mas o dígito mais bem reconhecido continua sendo o 1, provavelmente por seu formato simples e singular, enquanto que o dígito 5 continua sendo o mais complexo para a máquina, frequentemente confundido com o 3, provavelmente por seu traçado passar por muitos pontos em comum.

Na figura 3,foi reconstruído um dígito 5 que foi reconhecido como 3, provavelmente por seu formato distorcido. Na figura 4, um dígito 3 reconhecido como 5, provavelmente por suas curvas acentuadas e distorcidas.

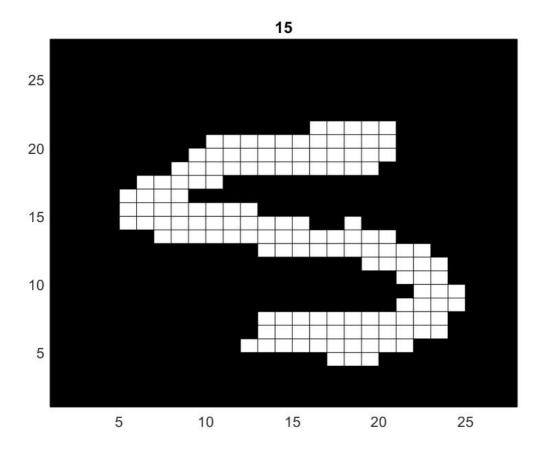


Figura 3: Um dígito cinco que foi reconhecido como três.

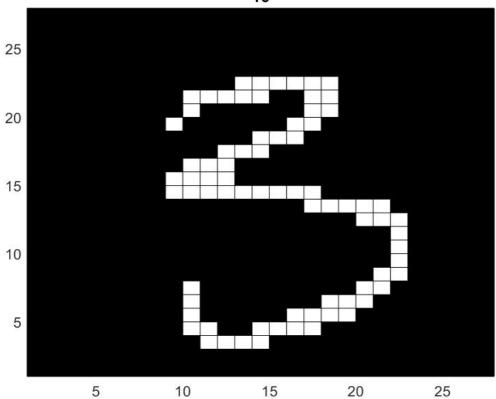


Figura 4: Um dígito três que foi reconhecido como cinco.

Q2.2) O ganho de desempenho se deve ao mapeamento com a função de tangente hiperbólica, que permite a saturação da saída do neurônio e evita que poucas entradas bem alinhadas, porém de magnitude alta, induzam um resultado incorreto. Assim, ficam sendo mais influentes as entradas com muitos pontos coincidentes aos maiores valores dos classificadores, em detrimento daquelas que possuem poucos pontos de grande valor sendo multiplicadas pelos elementos do respectivo classificador.

A diminuição do tempo de execução se deve ao fato de que a matriz de entrada e a matriz W reduziram de 784 para 500 (sem considerar o offset) seu número de colunas e linhas, respectivamente. Com isso, a quantidade de operações diminui significativamente, pois a quantidade de elementos a serem operados nestas matrizes é grande e seu número foi reduzido em pouco mais de 1 terço.

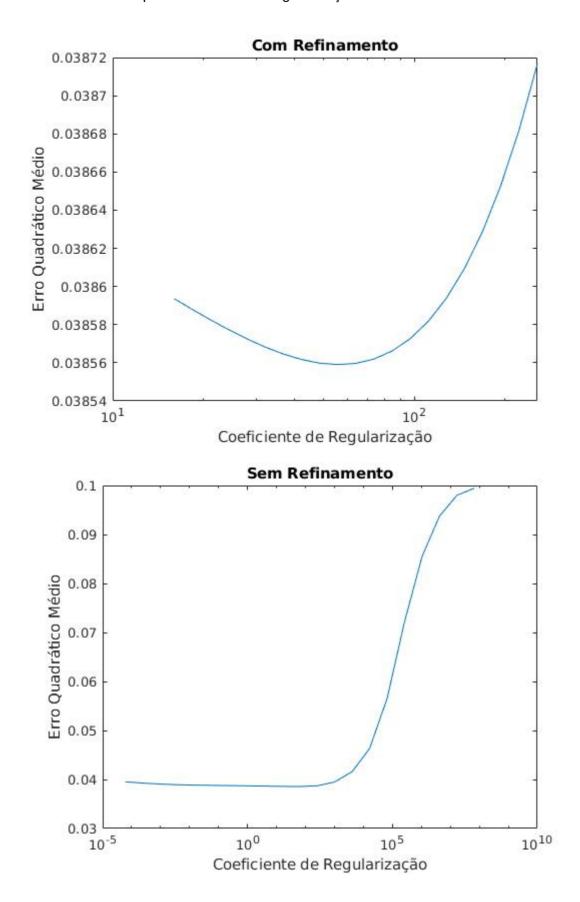
Q2.3) Como a matriz H é produzida através da multiplicação entre a entrada X e a matriz V de números normalmente distribuídos, o posto de H é maior que o de X, o que faz com que a necessidade de regularização no caso da ELM seja menor. Como calculamos a regularização de forma a penalizar o crescimento dos vetores de W, o coeficiente de regularização será o menor possível para realizar a tarefa. Portanto, o coeficiente de regularização utilizado no classificador linear (675.588) foi maior do que para a ELM (64.000).

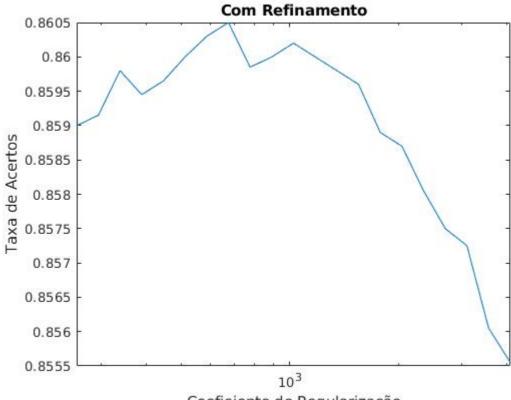
Q2.4) Como a matriz de entrada será diferente para cada caso dos pesos aleatórios, o

coeficiente de regularização pode ser alterado, para mais ou para menos, dentro de uma faixa de valores que é relativa ao desvio padrão escolhido para a distribuição normal dos pesos aleatórios.

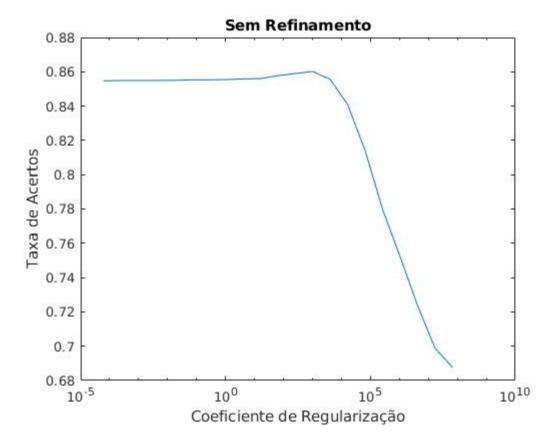
Gráficos de busca do coeficiente de regularização no anexo 3.

Anexo 1 - Busca pelo coeficiente de regularização da atividade

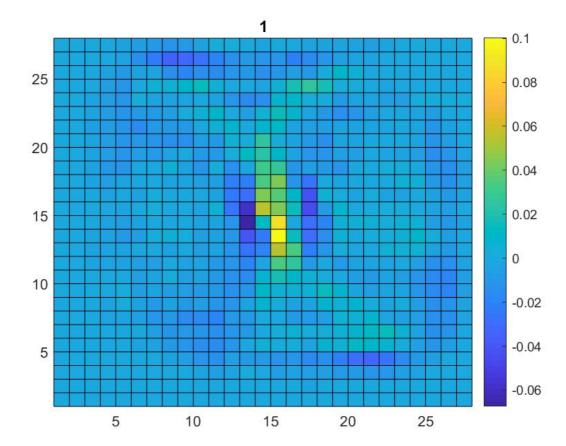


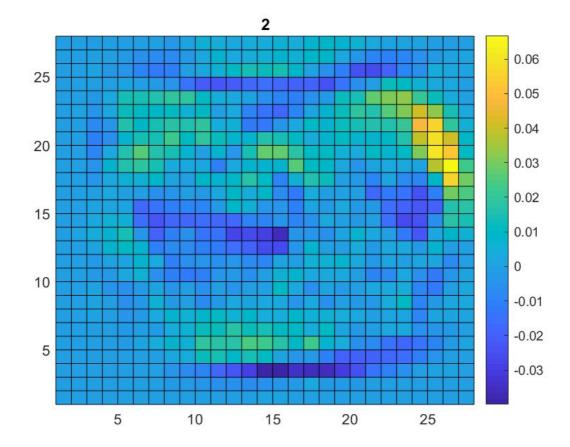


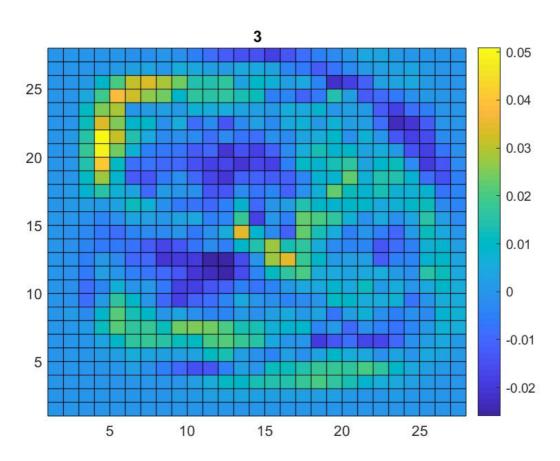
Coeficiente de Regularização

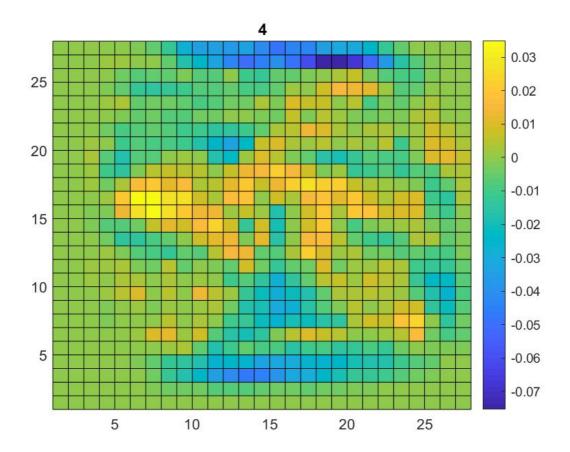


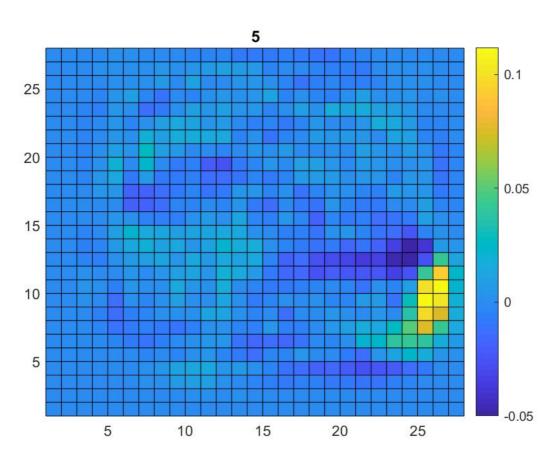
Anexo 2 - Gráficos de calor da atividade 1

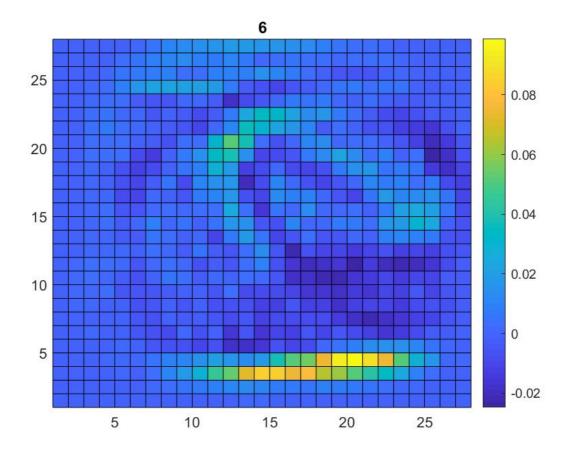


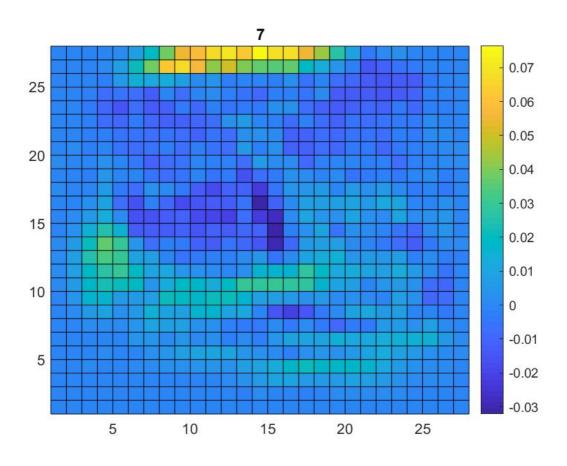


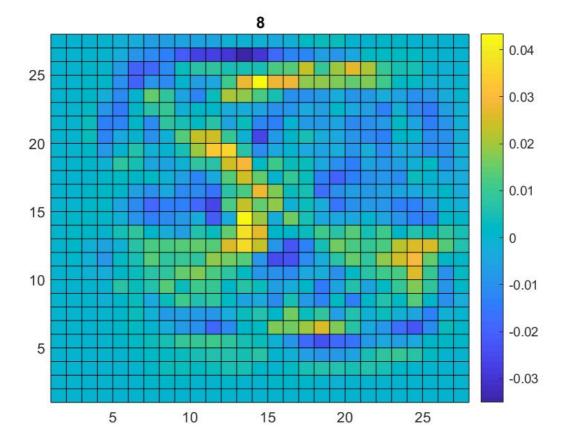


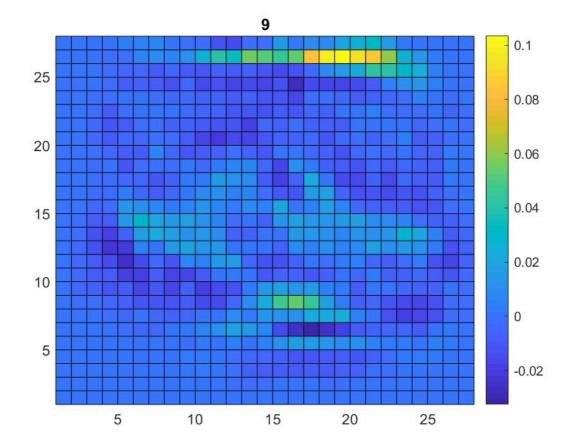


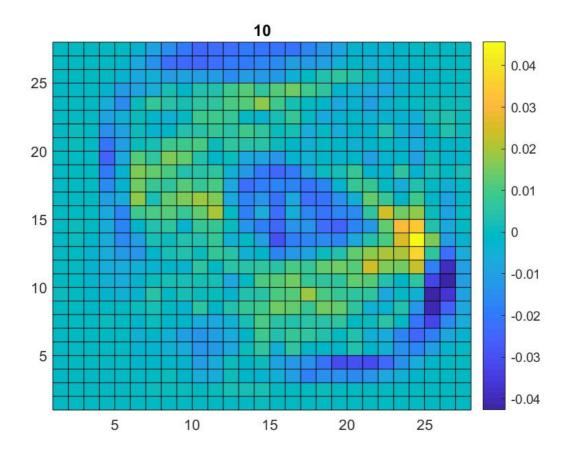




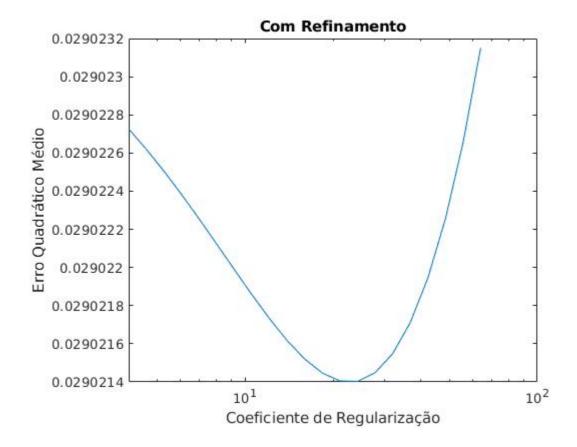


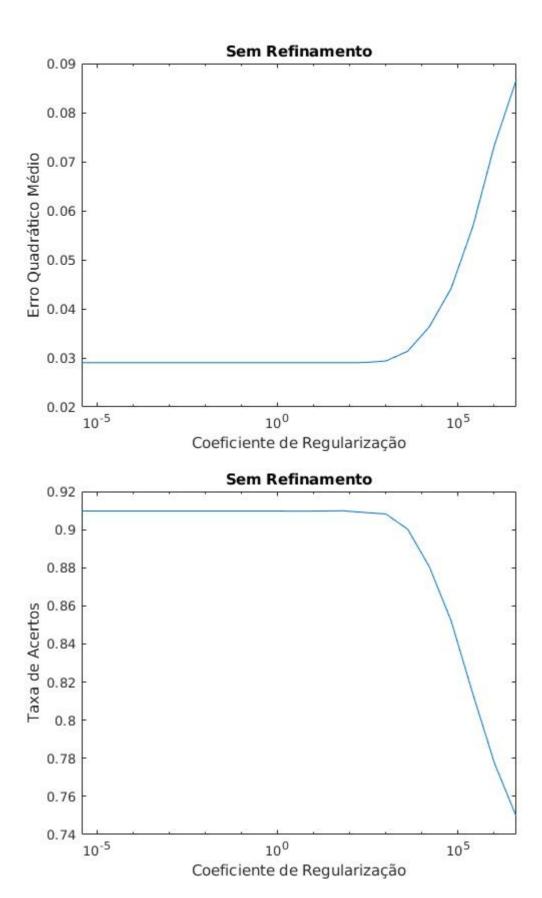


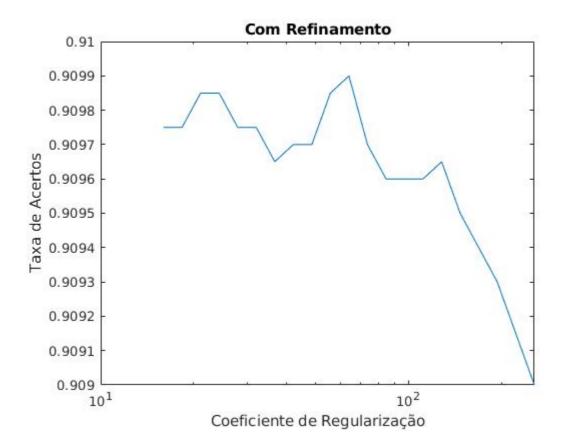




Anexo 3 - Busca pelo coeficiente de regularização da atividade 2







Patrick de Carvalho Tavares Rezende Ferreira - RA: 175480 - EFC1

Repositório: https://github.com/patrickctrf/EA072-Inteligencia-Artificial-IA/tree/master/EFC1 https://github.com/patrickctrf/EA072-Inteligencia-Artificial-IA/tree/master/EFC1

In [0]:

%cd /content/drive/My\ Drive/PODE\ APAGAR/EA072-EF1

/content/drive/My Drive/PODE APAGAR/EA072-EF1

Questão 3

Inicialmente, executou-se 5 vezes o código sugerido inicial a fim de verificar o desempenho da proposta. Seu desempenho foi de:

Loss: 0.0733; Acurácia: 0.9775.

Utilizando-se o método de tentativa e erro, foi criado um script que verificava o desempenho da rede para diferentes parâmetros alterados como dropout (0.1 a 0.6), número de camadas (1 a 4 intermediárias), épocas de treinamento (4 a 8) e número de neurônios por camada (256 a 512). O script executava esta mudança de parâmetros dentro de loops "for" para executar todas as combinações possíveis e tirava também a média das múltiplas execuções com mesmos parâmetros, a fim de se obter uma média de desempenho mais confiável. Os resultados desta varredura eram salvos ao final das execuções em um arquivo "listas.txt", permitindo ao usuários verificar qual a configuração obteve melhor desempenho. Foram utilizadas 4 threads - para varredura de redes de 1 a 4 camadas - durante o treinamento, a fim de promover paralelismo e diminuir o tempo requerido, que chegava a dezenas de horas.

Analisando as configurações que obtiveram o melhor desempenho, pode-se notar que as características que o maximizavam eram: 2 camadas, 512 neurônios, taxa de dropout próxima de 0.4 e 8 épocas de treinamento.

Portanto, para a proposta final deste modelo, foi executada mais um treinamento com uso dos atributos analisados acima e os parâmetros que resultaram no melhor desempenho durante a varredura foram:

• Camadas: 2; Neurônios por camada: 512; Dropout: 0.4; Épocas: 8.

O desempenho médio obtido foi de:

Loss: 0.0663; Acurácia: 0.9823.

Ambas as soluções consumiram um tempo de execução da ordem de poucos minutos e a diferença de desempenho foi cerca de 0,5% em ganho.

Os arquivos utilizados foram (no diretório q3):

Proposta Inicial: q3Inicial.py

Script de Varredura de parâmetros: q3.py

Proposta Final: q3Final.py

```
In [0]:
# q3Inicial.py
import tensorflow as tf
import os
mnist = tf.keras.datasets.mnist
(x train, y train),(x test, y test) = mnist.load data()
x train, x test = x train / 255.0, x test / 255.0
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation=tf.nn.relu),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
loss='sparse categorical crossentropy',
metrics=['accuracy'])
model.fit(x train, y train, epochs=5)
model.evaluate(x test, y test)
model json = model.to json()
json_file = open("model_MLP.json", "w")
json file.write(model json)
json file.close()
model.save weights("model MLP.h5")
print("Model saved to disk")
os.getcwd()
Epoch 1/5
60000/60000 [=========== ] - 6s 103us/sample - los
s: 0.2698 - acc: 0.9197
Epoch 2/5
s: 0.1385 - acc: 0.9579
Epoch 3/5
```

```
60000/60000 [============ ] - 6s 103us/sample - los
s: 0.1080 - acc: 0.9668
Epoch 4/5
60000/60000 [============ ] - 6s 101us/sample - los
s: 0.0928 - acc: 0.9710
Epoch 5/5
60000/60000 [===============] - 6s 101us/sample - los
s: 0.0821 - acc: 0.9736
10000/10000 [============ ] - 1s 68us/sample - los
s: 0.0640 - acc: 0.9797
Model saved to disk
```

Out[0]:

^{&#}x27;/content/drive/My Drive/PODE APAGAR/EA072-EF1'

In [0]:

```
#q3.py
import tensorflow as tf
import os
import threading
myMutex = threading.Lock()
value = "teste"
numeroDeNeuronios = []
numeroDeEpocas = []
numeroDeCamadas = []
numeroDeDropout = []
taxaDeAcertos = []
# Vamos colocar uma thread para treinar cada rede com um numero especifico de ca
madas.
def thread1Camadas(camadas):
        # Para tirar a media das iteracoes, somaremos todas aqui e dividiremos p
elo total.
        somaDasEficienciasDeCadaIteracao = 0
        # Os valores que utilizaremos para dropout variarao de 10% a 90% (instru
        valoresDropout = range(10, 60, 10)# Variaremos de 10% em 10%.
        valoresDropout = [i/100 for i in valoresDropout]# Converte de porcentage
m para escala de 0 a 1.
        # Testando resultados com diferentes quantidades de epocas.
        for epocas in [8, 4]:
                # Testando resultados com diferentes quantidades de neuronios.
                for neuronios in [256, 512]:
                        # So para indicar em que passo da execucao estamos.
                        print("\n\nepocas: " + str(epocas) + "\nCAMADAS" + str(c
amadas) + ": " + str(neuronios) + "\n\n")
                        # Este loop fica responsável por treinar com diferentes
 taxas de dropout.
                        # "i" eh o valor a cada iteracao.
                        for taxaDropout in valoresDropout:
                                # Repetimos o treinamento algumas vezes para tir
ar uma media da eficiencia
                                for iteracaoMedia in range(1,4):
                                         mnist = tf.keras.datasets.mnist
                                         (x_train, y_train),(x_test, y_test) = mn
ist.load_data()
                                         x_{train}, x_{test} = x_{train} / 255.0, x_{test}
t / 255.0
                                         model = tf.keras.models.Sequential([
                                          tf.keras.layers.Flatten(),
                                          tf.keras.layers.Dense(neuronios, activa
tion=tf.nn.relu),
                                          tf.keras.layers.Dropout(taxaDropout),#
Diferentes valores de dropout.
                                          tf.keras.layers.Dense(10, activation=tf
```

```
.nn.softmax)
                                         1)
                                         model.compile(optimizer='adam',
                                         loss='sparse_categorical_crossentropy',
                                          metrics=['accuracy'])
                                         model.fit(x train, y train, epochs=epoca
s)
                                         value = model.evaluate(x test, y test)
                                         model json = model.to json()
                                         json file = open("model MLP1.json", "w")
                                         json file.write(model json)
                                         json file.close()
                                         model.save weights("model MLP1.h5")
                                         print("Model saved to disk")
                                         os.getcwd()
                                         somaDasEficienciasDeCadaIteracao = value
[1] + somaDasEficienciasDeCadaIteracao
                                myMutex.acquire()
                                numeroDeNeuronios.append(neuronios)
                                numeroDeEpocas.append(epocas)
                                numeroDeCamadas.append(camadas)
                                numeroDeDropout.append(taxaDropout)
                                taxa De Acertos.append (soma Das Eficiencias De Cada Ite
racao/iteracaoMedia)
                                myMutex.release()
                                # Reiniciamos a soma.
                                somaDasEficienciasDeCadaIteracao = 0
# Vamos colocar uma thread para treinar cada rede com um numero especifico de ca
def thread2Camadas(camadas):
        # Para tirar a media das iteracoes, somaremos todas aqui e dividiremos p
elo total.
        somaDasEficienciasDeCadaIteracao = 0
        # Os valores que utilizaremos para dropout variarao de 10% a 90% (instru
cao abaixo).
        valoresDropout = range(10, 60, 10)# Variaremos de 10% em 10%.
        valoresDropout = [i/100 for i in valoresDropout]# Converte de porcentage
m para escala de 0 a 1.
        # Testando resultados com diferentes quantidades de epocas.
        for epocas in [8, 4]:
                # Testando resultados com diferentes quantidades de neuronios.
                for neuronios in [256, 512]:
                        # So para indicar em que passo da execucao estamos.
                        print("\n\nepocas: " + str(epocas) + "\nCAMADAS" + str(c
amadas) + ": " + str(neuronios) + "\n\n")
                        # Este loop fica responsável por treinar com diferentes
 taxas de dropout.
                        # "i" eh o valor a cada iteracao.
                        for taxaDropout in valoresDropout:
```

```
# Repetimos o treinamento algumas vezes para tir
ar uma media da eficiencia
                                 for iteracaoMedia in range(1,4):
                                         mnist = tf.keras.datasets.mnist
                                         (x train, y train),(x test, y test) = mn
ist.load data()
                                         x_{train}, x_{test} = x_{train} / 255.0, x_{test}
t / 255.0
                                         model = tf.keras.models.Sequential([
                                          tf.keras.layers.Flatten(),
                                          tf.keras.layers.Dense(neuronios, activa
tion=tf.nn.relu),
                                          tf.keras.layers.Dropout(taxaDropout),#
Diferentes valores de dropout.
                                          tf.keras.layers.Dense(neuronios, activa
tion=tf.nn.relu).
                                          tf.keras.layers.Dropout(taxaDropout),#
Diferentes valores de dropout.
                                          tf.keras.layers.Dense(10, activation=tf
.nn.softmax)
                                         1)
                                         model.compile(optimizer='adam',
                                          loss='sparse categorical crossentropy',
                                          metrics=['accuracy'])
                                         model.fit(x_train, y_train, epochs=epoca
s)
                                         value = model.evaluate(x test, y test)
                                         model json = model.to json()
                                         json file = open("model MLP2.json", "w")
                                         json file.write(model json)
                                         json file.close()
                                         model.save weights("model MLP2.h5")
                                         print("Model saved to disk")
                                         os.getcwd()
                                         somaDasEficienciasDeCadaIteracao = value
[1] + somaDasEficienciasDeCadaIteracao
                                 myMutex.acquire()
                                numeroDeNeuronios.append(neuronios)
                                 numeroDeEpocas.append(epocas)
                                numeroDeCamadas.append(camadas)
                                 numeroDeDropout.append(taxaDropout)
                                 taxaDeAcertos.append(somaDasEficienciasDeCadaIte
racao/iteracaoMedia)
                                myMutex.release()
                                 # Reiniciamos a soma.
                                 somaDasEficienciasDeCadaIteracao = 0
# Vamos colocar uma thread para treinar cada rede com um numero especifico de ca
madas.
def thread3Camadas(camadas):
        # Para tirar a media das iteracoes, somaremos todas aqui e dividiremos p
elo total.
        somaDasEficienciasDeCadaIteracao = 0
```

```
# Os valores que utilizaremos para dropout variarao de 10% a 90% (instru
cao abaixo).
        valoresDropout = range(10, 60, 10)# Variaremos de 10% em 10%.
        valoresDropout = [i/100 for i in valoresDropout]# Converte de porcentage
m para escala de 0 a 1.
        # Testando resultados com diferentes quantidades de epocas.
        for epocas in [8, 4]:
                # Testando resultados com diferentes quantidades de neuronios.
                for neuronios in [256, 512]:
                        # So para indicar em que passo da execucao estamos.
                        print("\n\nepocas: " + str(epocas) + "\nCAMADAS" + str(c
amadas) + ": " + str(neuronios) + "\n\n")
                        # Este loop fica responsável por treinar com diferentes
 taxas de dropout.
                        # "i" eh o valor a cada iteracao.
                        for taxaDropout in valoresDropout:
                                # Repetimos o treinamento algumas vezes para tir
ar uma media da eficiencia
                                for iteracaoMedia in range(1,4):
                                        mnist = tf.keras.datasets.mnist
                                        (x_train, y_train),(x_test, y_test) = mn
ist.load data()
                                        x train, x test = x train / 255.0, x tes
t / 255.0
                                        model = tf.keras.models.Sequential([
                                         tf.keras.layers.Flatten(),
                                         tf.keras.layers.Dense(neuronios, activa
tion=tf.nn.relu),
                                         tf.keras.layers.Dropout(taxaDropout),#
Diferentes valores de dropout.
                                         tf.keras.layers.Dense(neuronios, activa
tion=tf.nn.relu),
                                         tf.keras.layers.Dropout(taxaDropout),#
Diferentes valores de dropout.
                                         tf.keras.layers.Dense(neuronios, activa
tion=tf.nn.relu),
                                         tf.keras.layers.Dropout(taxaDropout),#
Diferentes valores de dropout.
                                         tf.keras.layers.Dense(10, activation=tf
.nn.softmax)
                                        1)
                                        model.compile(optimizer='adam',
                                         loss='sparse_categorical_crossentropy',
                                         metrics=['accuracy'])
                                        model.fit(x_train, y_train, epochs=epoca
s)
                                        value = model.evaluate(x test, y test)
                                        model json = model.to json()
                                        json_file = open("model_MLP3.json", "w")
                                        json file.write(model json)
                                        json_file.close()
                                        model.save weights("model MLP3.h5")
                                        print("Model saved to disk")
                                        os.getcwd()
                                        somaDasEficienciasDeCadaIteracao = value
```

```
[1] + somaDasEficienciasDeCadaIteracao
                                myMutex.acquire()
                                numeroDeNeuronios.append(neuronios)
                                numeroDeEpocas.append(epocas)
                                numeroDeCamadas.append(camadas)
                                numeroDeDropout.append(taxaDropout)
                                taxaDeAcertos.append(somaDasEficienciasDeCadaIte
racao/iteracaoMedia)
                                myMutex.release()
                                # Reiniciamos a soma.
                                somaDasEficienciasDeCadaIteracao = 0
# Vamos colocar uma thread para treinar cada rede com um numero especifico de ca
madas.
def thread4Camadas(camadas):
        # Para tirar a media das iteracoes, somaremos todas aqui e dividiremos p
elo total.
        somaDasEficienciasDeCadaIteracao = 0
        # Os valores que utilizaremos para dropout variarao de 10% a 90% (instru
cao abaixo).
        valoresDropout = range(10, 60, 10)# Variaremos de 10% em 10%.
        valoresDropout = [i/100 for i in valoresDropout]# Converte de porcentage
m para escala de 0 a 1.
        # Testando resultados com diferentes quantidades de epocas.
        for epocas in [8, 4]:
                # Testando resultados com diferentes quantidades de neuronios.
                for neuronios in [256, 512]:
                        # So para indicar em que passo da execucao estamos.
                        print("\n\nepocas: " + str(epocas) + "\nCAMADAS" + str(c
amadas) + ": " + str(neuronios) + "\n\n")
                        # Este loop fica responsável por treinar com diferentes
 taxas de dropout.
                        # "i" eh o valor a cada iteracao.
                        for taxaDropout in valoresDropout:
                                # Repetimos o treinamento algumas vezes para tir
ar uma media da eficiencia
                                for iteracaoMedia in range(1,4):
                                         mnist = tf.keras.datasets.mnist
                                         (x_train, y_train),(x_test, y_test) = mn
ist.load data()
                                         x_{train}, x_{test} = x_{train} / 255.0, x_{test}
t / 255.0
                                         model = tf.keras.models.Sequential([
                                         tf.keras.layers.Flatten(),
                                         tf.keras.layers.Dense(neuronios, activa
tion=tf.nn.relu),
                                         tf.keras.layers.Dropout(taxaDropout),#
Diferentes valores de dropout.
                                         tf.keras.layers.Dense(neuronios, activa
tion=tf.nn.relu),
```

```
tf.keras.layers.Dropout(taxaDropout),#
Diferentes valores de dropout.
                                         tf.keras.layers.Dense(neuronios, activa
tion=tf.nn.relu),
                                         tf.keras.layers.Dropout(taxaDropout),#
Diferentes valores de dropout.
                                         tf.keras.layers.Dense(neuronios, activa
tion=tf.nn.relu),
                                         tf.keras.layers.Dropout(taxaDropout),#
Diferentes valores de dropout.
                                         tf.keras.layers.Dense(10, activation=tf
.nn.softmax)
                                        1)
                                        model.compile(optimizer='adam',
                                         loss='sparse categorical crossentropy',
                                         metrics=['accuracy'])
                                        model.fit(x train, y train, epochs=epoca
s)
                                        value = model.evaluate(x test, y test)
                                        model json = model.to json()
                                        json file = open("model MLP4.json", "w")
                                        json file.write(model json)
                                        json_file.close()
                                        model.save weights("model MLP4.h5")
                                        print("Model saved to disk")
                                        os.getcwd()
                                        somaDasEficienciasDeCadaIteracao = value
[1] + somaDasEficienciasDeCadaIteracao
                                myMutex.acquire()
                                numeroDeNeuronios.append(neuronios)
                                numeroDeEpocas.append(epocas)
                                numeroDeCamadas.append(camadas)
                                numeroDeDropout.append(taxaDropout)
                                taxaDeAcertos.append(somaDasEficienciasDeCadaIte
racao/iteracaoMedia)
                                myMutex.release()
                                # Reiniciamos a soma.
                                somaDasEficienciasDeCadaIteracao = 0
if name == ' main ':
        camadas1 = threading.Thread(target=thread1Camadas,args=(1,))
        camadas2 = threading.Thread(target=thread2Camadas,args=(2,))
        camadas3 = threading.Thread(target=thread3Camadas,args=(3,))
        camadas4 = threading.Thread(target=thread4Camadas,args=(4,))
        camadas1.start()
        camadas2.start()
        camadas3.start()
        camadas4.start()
        try:
                camadas4.join();
```

```
except:
        pass;
try:
        camadas3.join();
except:
        pass;
try:
        camadas2.join();
except:
        pass;
try:
        camadas1.join();
except:
        pass;
listasFile = open("listasFFULLYCONNECTED.txt", "w")
listasFile.write(str(numeroDeNeuronios) + "\n")
listasFile.write(str(numeroDeEpocas) + "\n")
listasFile.write(str(numeroDeCamadas) + "\n")
listasFile.write(str(numeroDeDropout) + "\n")
listasFile.write(str(taxaDeAcertos) + "\n")
listasFile.close()
```

```
epocas: 8
CAMADAS1: 256
epocas: 8
CAMADAS2: 256
epocas: 8
CAMADAS3: 256
epocas: 8
CAMADAS4: 256
Epoch 1/8
6112/60000 [==>.....] - ETA: 12s - loss: 0.
6049 - acc: 0.8253Epoch 1/8
Epoch 1/8 1760/60000 [.....] - ETA: 32s -
loss: 1.0388 - acc: 0.7000
12608/60000 [=====>.....] - ETA: 11s - loss: 0.
4459 - acc: 0.8698
60000/60000 [=========== ] - 23s 379us/sample -
loss: 0.2369 - acc: 0.9305
Epoch 2/8
60000/60000 [============] - 26s 434us/sample -
loss: 0.2229 - acc: 0.9330
poch 2/8
7616/60000 [==>.....]60000/60000 [=======
9s 476us/sample - loss: 0.2287 - acc: 0.9308
Epoch 2/8
60000/60000 [==========] 3328/60000
[>.....] - 30s 492us/sample - loss: 0.2444
- acc: 0.9270
- ETA: 26s - loss: 0.1149 - acc: 0.9645Epoch 2/8
60000/60000 [=========== ] - 25s 419us/sample -
loss: 0.1021 - acc: 0.9693
Epoch 3/8
loss: 0.1005 - acc: 0.9692
Epoch 3/8
60000/60000 [==============] - 28s 463us/sample -
loss: 0.1098 - acc: 0.9671
loss: 0.1237 - acc: 0.9637
18080/60000 [======>.....] - ETA: 18s - loss: 0.
0700 - acc: 0.9779Epoch 3/8
60000/60000 [=========== ] - 25s 418us/sample -
loss: 0.0715 - acc: 0.9772
30496/60000 [=========>.....] - ETA: 13s - loss: 0.
```

```
0825 - acc: 0.9740Epoch 4/
loss: 0.0723 - acc: 0.9771
Epoch 4/
60000/60000 [=========== ] - 28s 465us/sample -
loss: 0.0835 - acc: 0.9741
33120/60000 [========>.....] - ETA: 11s - loss: 0.
loss: 0.0930 - acc: 0.9727
23776/60000 [======>.....] - ETA: 16s - loss: 0.
0547 - acc: 0.9823Epoch 4/8
loss: 0.0541 - acc: 0.9831
24288/60000 [=======>.....] - ETA: 16s - loss: 0.
0631 - acc: 0.9804Epoch 5/8
60000/60000 [=========== ] - 26s 435us/sample -
loss: 0.0598 - acc: 0.9807
Epoch 5/8
loss: 0.0678 - acc: 0.9792
18240/60000 [======>.....] - ETA: 18s - loss: 0.
0460 - acc: 0.9859Epoch 5/8
11264/60000 [====>.....]60000/60000 [=======
9s 480us/sample - loss: 0.0786 - acc: 0.9771
Epoch 5/8
60000/60000 [=========== ] - 25s 419us/sample -
loss: 0.0434 - acc: 0.9857
Epoch 6/8
loss: 0.0512 - acc: 0.9842
Epoch 6/8
60000/60000 [=========== ] - 28s 460us/sample -
loss: 0.0594 - acc: 0.9818
Epoch 6/8
loss: 0.0688 - acc: 0.9795
14272/60000 [=====>.....] - ETA: 21s - loss: 0.
0422 - acc: 0.987
0
Epoch 6/8
loss: 0.0354 - acc: 0.9887
Epoch 7/8
60000/60000 [=========== ] - 27s 444us/sample -
loss: 0.0441 - acc: 0.9862
23136/60000 [======>.....] - ETA: 17s - loss: 0.
0584 - acc: 0.9835Epoch 7/8
60000/60000 [=========== ] - 27s 458us/sample -
loss: 0.0501 - acc: 0.9847
Epoch 7/8
60000/60000 [=========== ] - 25s 418us/sample -
loss: 0.0291 - acc: 0.9901
9792/60000 [===>.....] - ETA: 23s - loss: 0.
0360 - acc: 0.9868Epoch 8/8
60000/60000 [=========== ] - 29s 480us/sample -
```

```
loss: 0.0607 - acc: 0.9822
Epoch 7/8
60000/60000 [=========== ] - 26s 436us/sample -
loss: 0.0372 - acc: 0.9876
17888/60000 [======>.....] - ETA: 20s - loss: 0.
0491 - acc: 0.9861Epoch 8/8
40992/60000 [==========>.....]60000/60000 [=======
s 458us/sample - loss: 0.0460 - acc: 0.9851
252 - acc: 0.9915Epoch 8/8
60000/60000 [=========== ] - 25s 416us/sample -
loss: 0.0255 - acc: 0.9914
10000/10000 [============= ] - 4s 357us/sample - l
oss: 0.0689 - acc: 0.9810
53280/60000 [=============>....] - ETA: 3s - loss: 0.0
540 - acc: 0.9842Model saved to disk
18464/60000 [======>.....] - ETA: 19s - loss: 0.
0352 - acc: 0.9888Epoch 1/8
loss: 0.0531 - acc: 0.9845
Epoch 8/
loss: 0.0316 - acc: 0.9900
10000/10000 [============= ] - 4s 378us/sample - l
oss: 0.0821 - acc: 0.9771
41056/60000 [==========>.....] - ETA: 8s - loss: 0.0
379 - acc: 0.9877Model saved to disk
32800/60000 [=========>.....] - ETA: 11s - loss: 0.
3014 - acc: 0.9135Epoch 1/8
60000/60000 [============ ] - 27s 450us/sample -
loss: 0.0393 - acc: 0.9873
>.....] - 4s 354us/sample - loss: 0.1359 - acc: 0.9
697
466 - acc: 0.9859Model saved to disk
loss: 0.2339 - acc: 0.9325
Epoch 2/8
30048/60000 [========>.....] - ETA: 12s - loss: 0.
2928 - acc: 0.9117Epoch 1/8
60000/60000 [=========== ] - 27s 455us/sample -
loss: 0.0479 - acc: 0.9853
oss: 0.0765 - acc: 0.9795
43808/60000 [=============>.....] - ETA: 6s - loss: 0.2
515 - acc: 0.9236Model saved to disk
22080/60000 [=======>.....] - ETA: 17s - loss: 0.
3441 - acc: 0.8946Epoch 1/8
loss: 0.2205 - acc: 0.9327
37664/60000 [===========>.....] - ETA: 8s - loss: 0.1
071 - acc: 0.9688Epoch 2/8
60000/60000 [=========== ] - 24s 392us/sample -
loss: 0.1023 - acc: 0.9699
22912/60000 [======>.....] - ETA: 18s - loss: 0.
3744 - acc: 0.8853Epoch 3/8
60000/60000 [=========== ] - 27s 453us/sample -
```

```
loss: 0.2291 - acc: 0.9299
Epoch 2/8
60000/60000 [=========== ] - 29s 482us/sample -
loss: 0.2523 - acc: 0.9250
Epoch 2/8
60000/60000 [===========] - 27s 448us/sample -
loss: 0.0977 - acc: 0.9696
25760/60000 [=======>.....] - ETA: 15s - loss: 0.
loss: 0.0708 - acc: 0.9781
Epoch 4/8
loss: 0.1139 - acc: 0.9657
33440/60000 [=========>.....] - ETA: 12s - loss: 0.
1237 - acc: 0.9644Epoch 3/
60000/60000 [============ ] - 26s 437us/sample -
loss: 0.0699 - acc: 0.9777
23008/60000 [======>.....] - ETA: 16s - loss: 0.
0831 - acc: 0.9743Epoch 4/8
60000/60000 [============ ] - 29s 479us/sample -
loss: 0.1194 - acc: 0.9653
Epoch 3/8
loss: 0.0543 - acc: 0.9830
8032/60000 [===>.....] - ETA: 24s - loss: 0.
0825 - acc: 0.9747Epoch 5/8
loss: 0.0853 - acc: 0.9740
25888/60000 [=======>.....] - ETA: 14s - loss: 0.
0406 - acc: 0.9876Epoch 4/8
loss: 0.0588 - acc: 0.9809
loss: 0.0938 - acc: 0.9725
428 - acc: 0.9867Epoch 4/8
60000/60000 [============= ] - 25s 421us/sample -
loss: 0.0427 - acc: 0.9868
 32/60000 [.....] - ETA: 44s - loss: 0.
0718 - acc: 0.9375Epoch 6/8
loss: 0.0693 - acc: 0.9793
29280/60000 [=======>....] - ETA: 14s - loss: 0.
0787 - acc: 0.9760Epoch 5/8
60000/60000 [=========== ] - 27s 446us/sample -
loss: 0.0467 - acc: 0.9851
Epoch 6/8
loss: 0.0354 - acc: 0.9883
Epoch 7/8
60000/60000 [=========== ] - 28s 474us/sample -
loss: 0.0791 - acc: 0.9762
14560/60000 [=====>.....] - ETA: 19s - loss: 0.
0361 - acc: 0.9876Epoch 5/8
60000/60000 [============= ] - 28s 465us/sample -
loss: 0.0605 - acc: 0.9809
```

```
Epoch 6/8
loss: 0.0426 - acc: 0.9862
Epoch 7/8
60000/60000 [============ ] - 24s 407us/sample -
loss: 0.0294 - acc: 0.9901
Epoch 8/8
60000/60000 [=========== ] - 29s 483us/sample -
loss: 0.0676 - acc: 0.9805
20800/60000 [======>.....] - ETA: 17s - loss: 0.
0353 - acc: 0.9887Epoch 6/8
60000/60000 [============ ] - 28s 466us/sample -
232 - acc: 0.9922Epoch 7/8
loss: 0.0236 - acc: 0.9921
0595 - acc: 0.9814 - 27s 447us/sample - loss: 0.0373 - acc: 0.9877
Epoch 8/8
10000/10000 [============= ] - 4s 375us/sample - l
oss: 0.0729 - acc: 0.9801
20000/60000 [=======>.....] - ETA: 18s - loss: 0.
0454 - acc: 0.9864Model saved to disk
53888/60000 [==============>....] - ETA: 3s - loss: 0.0
614 - acc: 0.9814Epoch 1/8
60000/60000 [============ ] - 30s 493us/sample -
loss: 0.0606 - acc: 0.9817
Epoch 7/8
60000/60000 [============ ] - 28s 464us/sample -
loss: 0.0472 - acc: 0.9858
31488/60000 [========>.....] - ETA: 12s - loss: 0.
3076 - acc: 0.9106Epoch 8/8
60000/60000 [============ ] - 27s 458us/sample -
loss: 0.0322 - acc: 0.9895
oss: 0.0723 - acc: 0.9808
19488/60000 [======>.....] - ETA: 19s - loss: 0.
0432 - acc: 0.9868Model saved to disk
60000/60000 [=========]24224/60000 [=======
- ETA: 16s - loss: 0.0442 - acc: 0.9870Epoch 2/8
543 - acc: 0.9839Epoch 1/8
loss: 0.0564 - acc: 0.9833
Epoch 8/8
60000/60000 [=========== ] - 28s 472us/sample -
loss: 0.0457 - acc: 0.9860
10000/10000 [============= ] - 4s 385us/sample - l
oss: 0.0767 - acc: 0.9823
29312/60000 [========>...... - ETA: 14s - loss: 0.
0480 - acc: 0.9863Model saved to disk
37056/60000 [==========>.....] - ETA: 10s - loss: 0.
0472 - acc: 0.9868Epoch 1/8
60000/60000 [=========== ] - 25s 417us/sample -
loss: 0.1027 - acc: 0.9683
A: 10s - loss: 0.0471 - acc: 0.9868Epoch 3/8
```

```
60000/60000 [===========] - 27s 447us/sample -
loss: 0.2203 - acc: 0.9327
>.....]
loss: 0.0504 - acc: 0.9856
10000/10000 [============== ] - 4s 387us/sample - l
oss: 0.0945 - acc: 0.9764
33888/60000 [========>.....]37728/60000 [=======
TA: 9s - loss: 0.0751 - acc: 0.9768Model saved to disk
729 - acc: 0.9776Epoch 1/8
60000/60000 [=========== ] - 24s 394us/sample -
loss: 0.0720 - acc: 0.9779
5888/60000 [=>.....] - ETA: 31s - loss: 0.
6878 - acc: 0.782
Epoch 4/8
loss: 0.2302 - acc: 0.9297
Epoch 2/8
loss: 0.0990 - acc: 0.9693
3328/60000 [>.....] - ETA: 27s - loss: 0.
0946 - acc: 0.9681Epoch 3/8
60000/60000 [=========== ] - 25s 418us/sample -
loss: 0.0553 - acc: 0.9824
Epoch 5/8
60000/60000 [=========== ] - 30s 495us/sample -
loss: 0.2513 - acc: 0.9234
2752/60000 [>.....] - ETA: 24s - loss: 0.
0492 - acc: 0.9847Epoch 2/8
60000/60000 [============ ] - 28s 467us/sample -
loss: 0.1078 - acc: 0.9675
loss: 0.0727 - acc: 0.9768
1952/60000 [.....] - ETA: 27s - loss: 0.
0908 - acc: 0.9672Epoch 4/8
loss: 0.0443 - acc: 0.9858
Epoch 6/8
60000/60000 [=========== ] - 30s 500us/sample -
loss: 0.1244 - acc: 0.9640
poch 3/852608/60000 [===========>....]
60000/60000 [=========== ] - 27s 457us/sample -
loss: 0.0570 - acc: 0.9818
Epoch 5/8
6848/60000 [==>.....]60000/60000 [=======
9s 477us/sample - loss: 0.0835 - acc: 0.9745
Epoch 4/8
loss: 0.0349 - acc: 0.9884
34400/60000 [==========>.....] - ETA: 12s - loss: 0.
0653 - acc: 0.9801Epoch 7/8
60000/60000 [=========== ] - 31s 509us/sample -
loss: 0.0957 - acc: 0.9722
```

```
59456/60000 [===========>.]Epoch 4/8
60000/60000 [=========== ] - 28s 461us/sample -
loss: 0.0496 - acc: 0.9834
Epoch 6/8
60000/60000 [=========== ] - 29s 484us/sample -
loss: 0.0686 - acc: 0.9792
Epoch 5/8
loss: 0.0294 - acc: 0.9903
Epoch 8/8
60000/60000 [=========== ] - 27s 457us/sample -
loss: 0.0410 - acc: 0.9865
54688/60000 [============>...]Epoch 7/8
loss: 0.0787 - acc: 0.9767
3616/60000 [>.....] - ETA: 24s - loss: 0.
0283 - acc: 0.9898Epoch 5/8
60000/60000 [============= ] 5568/60000 [=
>.....] - 28s 470us/sample - loss: 0.0604 -
acc: 0.9813
- ETA: 24s - loss: 0.0268 - acc: 0.9910Epoch 6/8
60000/60000 [=========== ] - 26s 431us/sample -
loss: 0.0253 - acc: 0.9917
10000/10000 [============== ] - 4s 375us/sample - l
oss: 0.0711 - acc: 0.9795
33440/60000 [========>.....] - ETA: 12s - loss: 0.
0685 - acc: 0.9797Model saved to disk
47136/60000 [============>.....] - ETA: 5s - loss: 0.0
341 - acc: 0.9886Epoch 1/8
loss: 0.0346 - acc: 0.9884
50752/60000 [=============>....] - ETA: 4s - loss: 0.0
loss: 0.0694 - acc: 0.9797
9984/60000 [===>.....] - ETA: 23s - loss: 0.
0244 - acc: 0.9918Epoch 6/8
loss: 0.2490 - acc: 0.9269
Epoch 2/8
60000/60000 [=========== ] - 28s 460us/sample -
loss: 0.0333 - acc: 0.9890
loss: 0.0455 - acc: 0.9859
Epoch 8/8
10000/10000 [============= ] - 4s 391us/sample - l
oss: 0.0731 - acc: 0.9803
ETA: 1s - loss: 0.0589 - acc: 0.9826
loss: 0.0593 - acc: 0.9825
Epoch 7/
4864/60000 [=>.....] - ETA: 24s - loss: 0.
0421 - acc: 0.9877Epoch 1/8
```

```
loss: 0.1132 - acc: 0.9664
Epoch 3/8
60000/60000 [=========== ] - 27s 458us/sample -
loss: 0.0408 - acc: 0.9875
60000/60000 [=========== ] - 29s 476us/sample -
loss: 0.0537 - acc: 0.9843
43232/60000 [===========>.....] - ETA: 6s - loss: 0.0
836 - acc: 0.9732Epoch 8/8
60000/60000 [============ ] - 27s 445us/sample -
loss: 0.2484 - acc: 0.9248
Epoch 2/8
10000/10000 [============== ] - 4s 371us/sample - l
oss: 0.0868 - acc: 0.9798
2656/60000 [>.....] - ETA: 30s - loss: 0.
0440 - acc: 0.9880Model saved to disk
11552/60000 [====>.....] - ETA: 20s - loss: 0.
1213 - acc: 0.9619Epoch 1/8
60000/60000 [============ ] - 24s 405us/sample -
loss: 0.0831 - acc: 0.9736
Epoch 4/8
60000/60000 [============ ] - 27s 447us/sample -
loss: 0.1159 - acc: 0.9639
Epoch 3/846304/60000 [==========>.....] - ETA: 6s -
loss: 0.2920 - acc: 0.9104
60000/60000 [=========== ] - 29s 484us/sample -
loss: 0.0479 - acc: 0.9863
60000/60000 [=========== ] - 25s 418us/sample -
loss: 0.0653 - acc: 0.9794
Epoch 5/8
oss: 0.0785 - acc: 0.9799
loss: 0.2632 - acc: 0.9195
14688/60000 [=====>.....] - ETA: 19s - loss: 0.
0876 - acc: 0.9732Epoch 2/8
15776/60000 [=====>.....] - ETA: 18s - loss: 0.
0869 - acc: 0.9730Model saved to disk
19200/60000 [======>....] - ETA: 14s - loss: 0.
0521 - acc: 0.9834Epoch 1/8
60000/60000 [============ ] - 25s 424us/sample -
loss: 0.0923 - acc: 0.9713
359 - acc: 0.9593Epoch 4/8
>.....] - 24s 403us/sample - loss: 0.0542 -
acc: 0.9826
- ETA: 25s - loss: 0.0616 - acc: 0.9802Epoch 6/8
loss: 0.1332 - acc: 0.9600
10432/60000 [====>.....] - ETA: 21s - loss: 0.
0445 - acc: 0.9868Epoch 3/8
17920/60000 [======>.....]60000/60000 [=======
Os 503us/sample - loss: 0.2882 - acc: 0.9119
30560/60000 [==========>.....] - ETA: 12s - loss: 0.
0452 - acc: 0.9858Epoch 2/8
60000/60000 [=========== ] - 27s 445us/sample -
loss: 0.0739 - acc: 0.9762
41408/60000 [==========>.....] - ETA: 8s - loss: 0.1
045 - acc: 0.9689Epoch 5/8
60000/60000 [===============] - 25s 418us/sample -
```

```
loss: 0.0454 - acc: 0.9850
Epoch 7/8
60000/60000 [=========] - 28s 459us/sample -
loss: 0.1046 - acc: 0.9689
Epoch 4/8
loss: 0.1479 - acc: 0.9578
Epoch 3/
18176/60000 [======>.....]60000/60000 [=======
7s 447us/sample - loss: 0.0637 - acc: 0.9798
Epoch 6/
loss: 0.0388 - acc: 0.9875
Epoch 8/8
882 - acc: 0.9736Buffered data was truncated after reaching the ou
tput size limit.
4
```

In [0]:

```
# q3Final.py
import tensorflow as tf
import os
mnist = tf.keras.datasets.mnist
(x train, y train),(x test, y test) = mnist.load data()
x_{train}, x_{test} = x_{train} / 255.0, x_{test} / 255.0
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation=tf.nn.relu),
tf.keras.layers.Dropout(0.3),
tf.keras.layers.Dense(512, activation=tf.nn.relu),
tf.keras.layers.Dropout(0.4),
tf.keras.layers.Dense(10, activation=tf.nn.softmax)
model.compile(optimizer='adam',
loss='sparse categorical crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=8)
model.evaluate(x_test, y_test)
model json = model.to json()
json file = open("model MLP.json", "w")
json file.write(model json)
json file.close()
model.save weights("model MLP.h5")
print("Model saved to disk")
os.getcwd()
```

```
Epoch 1/8
s: 0.2446 - acc: 0.9246
Epoch 2/8
s: 0.1283 - acc: 0.9617
Epoch 3/8
60000/60000 [=========== ] - 5s 82us/sample - los
s: 0.1008 - acc: 0.9691
Epoch 4/8
60000/60000 [============ ] - 5s 82us/sample - los
s: 0.0869 - acc: 0.9737
Epoch 5/8
60000/60000 [============ ] - 5s 83us/sample - los
s: 0.0779 - acc: 0.9765
Epoch 6/8
s: 0.0712 - acc: 0.9787
Epoch 7/8
s: 0.0653 - acc: 0.9803
Epoch 8/8
60000/60000 [============ ] - 5s 83us/sample - los
s: 0.0618 - acc: 0.9819
10000/10000 [============= ] - 1s 58us/sample - los
s: 0.0669 - acc: 0.9823
Model saved to disk
```

Out[0]:

'/content'

Questão 4

Inicialmente, executou-se 5 vezes o código sugerido inicial a fim de verificar o desempenho da proposta. Seu desempenho foi de:

Loss: 0.0260; Acurácia: 0.9909.

Utilizando-se o método de tentativa e erro, foi criado um script que verificava o desempenho da rede para diferentes parâmetros alterados como dropout (0.1 a 0.6), número de filtros (32 a 64), épocas de treinamento (2 a 6) e formato dos kernel utilizados (2x2 ou 3x3). O script executava esta mudança de parâmetros dentro de loops "for" para executar todas as combinações possíveis e tirava também a média das múltiplas execuções com mesmos parâmetros, a fim de se obter uma média de desempenho mais confiável. Os resultados desta varredura eram salvos ao final das execuções em um arquivo "listas.txt", permitindo ao usuários verificar qual a configuração obteve melhor desempenho. Foram utilizadas 4 threads - para varredura de redes de 1 a 4 camadas - durante o treinamento, a fim de promover paralelismo e diminuir o tempo requerido, que era ainda maior que o demandado para a questão 1. Verificou-se que com 6 épocas de treinamento o resultado era levemente melhorado, mas não siginficativamente. A variação das demais grandezas fazia o desempenho diminuir nos testes. Então, após a varredura, foi realizado mais uma tentativa de treinamento com adição de uma camada convolucional e dropout de 0.3, o que elevou os resultados e nos trouxe à proposta final de código para esta questão.

Através da varredura, foi possível perceber que as alterações que implicavam em aumento de desempenho eram: Maior número de filtros, 2 camadas convolucionais, taxa de dropout próxima de 0.3 e kernel 3x3 (com max pool 2x2).

Portanto, para a proposta final deste modelo, os parâmetros alterados que resultaram no melhor desempenho durante a varredura foram:

 Adição de duas camadaa convolucionais com kernel 3x3 (seguida de uma max pool em 2x2) após a saída da primeira layer de max pool; 8 épocas de treinamento. Camadas convolucionais todas com 512 filtros e dropout de 0,3. Os demais parâmetros foram mantidos por não apresentar vantagem média significativa.

O desempenho médio obtido foi de:

Loss: 0.0190; Acurácia: 0.9935.

Ambas as soluções consumiram um tempo de execução da ordem de poucos minutos e a diferença de desempenho foi cerca de 0,22% em ganho.

Os arquivos utilizados foram (no diretório q2):

Proposta Inicial: q4Inicial.py

Script de Varredura de parâmetros: q4.py

Proposta Final: q4Final.py

Comparação entre ELM, MLP e CNN

Desempenho:

ELM: 91,09%MLP: 98,23%

• CNN: 99,35%

Nota-se claramente que a CNN apresenta o melhor desempenho dentre as 3 melhores técnicas utilizadas. Porém, o processo de treinamento para otimização desta toma dezenas de horas, enquanto que a ELM requer apenas alguns minutos para ser ajustada e ficar cerca de 8% abaixo em desempenho. Logo, se

In [0]:

```
# q4Inicial.pv
import tensorflow as tf
import os
mnist = tf.keras.datasets.mnist
(x train, y train),(x test, y test) = mnist.load data()
# reshape to be [samples][width][height][pixels]
x train = x train.reshape(x train.shape[0], 28, 28, 1)
x_{test} = x_{test.reshape}(x_{test.shape}[0], 28, 28, 1)
x train, x test = x train / 255.0, x test / 255.0
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, kernel size=(3, 3),
 activation='relu',
input shape=(28, 28, 1))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
model.compile(optimizer='adam',
loss='sparse categorical crossentropy',
metrics=['accuracy'])
model.fit(x train, y train, epochs=5)
model.evaluate(x test, y test)
model_json = model.to_json()
json file = open("model CNN.json", "w")
json file.write(model json)
json file.close()
model.save weights("model CNN.h5")
print("Model saved to disk")
os.getcwd()
Epoch 1/5
60000/60000 [=====
                    ss: 0.2004 - acc: 0.9403
Epoch 2/5
60000/60000 [============ ] - 15s 256us/sample - lo
ss: 0.0851 - acc: 0.9741
Epoch 3/5
ss: 0.0628 - acc: 0.9809
Epoch 4/5
60000/60000 [===========] - 15s 252us/sample - lo
ss: 0.0532 - acc: 0.9837
Epoch 5/5
60000/60000 [==============] - 15s 249us/sample - lo
ss: 0.0461 - acc: 0.9857
10000/10000 [============= ] - 1s 114us/sample - los
s: 0.0294 - acc: 0.9906
Model saved to disk
```

Out[0]:

^{&#}x27;/content/drive/My Drive/PODE APAGAR/EA072-EF1'

In [0]:

```
# q4.py
import tensorflow as tf
import os
import threading
myMutex = threading.Lock()
value = "teste"
numeroDeNeuronios = []
numeroDeEpocas = []
numeroDeCamadas = []
numeroDeDropout = []
taxaDeAcertos = []
# Vamos colocar uma thread para treinar cada rede com um numero especifico de ca
madas.
def thread1Camadas(camadas):
        # Para tirar a media das iteracoes, somaremos todas aqui e dividiremos p
elo total.
        somaDasEficienciasDeCadaIteracao = 0
        # Os valores que utilizaremos para dropout variarao de 10% a 90% (instru
        valoresDropout = range(10, 40, 10)# Variaremos de 10% em 10%.
        valoresDropout = [i/100 for i in valoresDropout]# Converte de porcentage
m para escala de 0 a 1.
        # Testando resultados com diferentes quantidades de epocas.
        for epocas in [2, 6]:
                # Testando resultados com diferentes quantidades de filtros.
                for filtros in [32, 64]:
                        # So para indicar em que passo da execucao estamos.
                        print("\n\nepocas: " + str(epocas) + "\nCAMADAS" + str(c
amadas) + ": " + str(filtros) + "\n\n")
                        # Este loop fica responsável por treinar com diferentes
 taxas de dropout.
                        # "i" eh o valor a cada iteracao.
                        for taxaDropout in valoresDropout:
                                 # Repetimos o treinamento algumas vezes para tir
ar uma media da eficiencia
                                 for iteracaoMedia in range(1,3):
                                         mnist = tf.keras.datasets.mnist
                                         (x_train, y_train),(x_test, y_test) = mn
ist.load_data()
                                         # reshape to be [samples][width][height]
[pixels]
                                         x_train = x_train.reshape(x_train.shape[
0], 28, 28, 1)
                                         x_test = x_test.reshape(x_test.shape[0],
28, 28, 1)
                                         x_{train}, x_{test} = x_{train} / 255.0, x_{test}
t / 255.0
                                         model = tf.keras.models.Sequential()
```

```
model.add(tf.keras.layers.Conv2D(filtros
, kernel_size=(3, 3),
                                          activation='relu',
                                         input shape=(28, 28, 1)))
                                        model.add(tf.keras.layers.Conv2D(filtros
*2, (3, 3), activation='relu'))
                                        model.add(tf.keras.layers.MaxPooling2D(p
ool size=(2, 2)))
                                        model.add(tf.keras.layers.Dropout(taxaDr
opout))
                                         model.add(tf.keras.layers.Flatten())
                                        model.add(tf.keras.layers.Dense(128, act
ivation='relu'))
                                        model.add(tf.keras.layers.Dropout(taxaDr
opout))
                                        model.add(tf.keras.layers.Dense(10, acti
vation='softmax'))
                                        model.compile(optimizer='adam',
                                         loss='sparse categorical crossentropy',
                                         metrics=['accuracy'])
                                         model.fit(x train, y train, epochs=epoca
s)
                                         value = model.evaluate(x test, y test)
                                         model json = model.to json()
                                         json file = open("model CNN1.json", "w")
                                         json file.write(model json)
                                         json file.close()
                                        model.save weights("model CNN1.h5")
                                         print("Model saved to disk")
                                         os.getcwd()
                                         somaDasEficienciasDeCadaIteracao = value
[1] + somaDasEficienciasDeCadaIteracao
                                myMutex.acquire()
                                numeroDeNeuronios.append(filtros)
                                numeroDeEpocas.append(epocas)
                                numeroDeCamadas.append(camadas)
                                numeroDeDropout.append(taxaDropout)
                                taxaDeAcertos.append(somaDasEficienciasDeCadaIte
racao/iteracaoMedia)
                                myMutex.release()
                                # Reiniciamos a soma.
                                somaDasEficienciasDeCadaIteracao = 0
# Vamos colocar uma thread para treinar cada rede com um numero especifico de ca
madas.
def thread2Camadas(camadas):
        # Para tirar a media das iteracoes, somaremos todas aqui e dividiremos p
elo total.
        somaDasEficienciasDeCadaIteracao = 0
        # Os valores que utilizaremos para dropout variarao de 10% a 90% (instru
cao abaixo).
        valoresDropout = range(10, 40, 10)# Variaremos de 10% em 10%.
        valoresDropout = [i/100 for i in valoresDropout]# Converte de porcentage
m para escala de 0 a 1.
```

```
# Testando resultados com diferentes quantidades de epocas.
        for epocas in [2, 6]:
                # Testando resultados com diferentes quantidades de filtros.
                for filtros in [32, 64]:
                        # So para indicar em que passo da execucao estamos.
                        print("\n\nepocas: " + str(epocas) + "\nCAMADAS" + str(c
amadas) + ": " + str(filtros) + "\n\n")
                        # Este loop fica responsável por treinar com diferentes
 taxas de dropout.
                        # "i" eh o valor a cada iteracao.
                        for taxaDropout in valoresDropout:
                                # Repetimos o treinamento algumas vezes para tir
ar uma media da eficiencia
                                for iteracaoMedia in range(1,3):
                                         mnist = tf.keras.datasets.mnist
                                         (x train, y train),(x test, y test) = mn
ist.load data()
                                         # reshape to be [samples][width][height]
[pixels]
                                         x train = x train.reshape(x train.shape[
0], 28, 28, 1)
                                         x test = x test.reshape(x test.shape[0],
28, 28, 1)
                                         x_{train}, x_{test} = x_{train} / 255.0, x_{test}
t / 255.0
                                         model = tf.keras.models.Sequential()
                                         model.add(tf.keras.layers.Conv2D(filtros
, kernel size=(2, 2),
                                         activation='relu',
                                         input shape=(28, 28, 1))
                                         model.add(tf.keras.layers.Conv2D(filtros
*2, (3, 3), activation='relu'))
                                         model.add(tf.keras.layers.MaxPooling2D(p
ool size=(2, 2)))
                                         model.add(tf.keras.layers.Conv2D(filtros
*2, (3, 3), activation='relu'))
                                         model.add(tf.keras.layers.MaxPooling2D(p
ool size=(2, 2)))
                                         model.add(tf.keras.layers.Dropout(taxaDr
opout))
                                         model.add(tf.keras.layers.Flatten())
                                         model.add(tf.keras.layers.Dense(128, act
ivation='relu'))
                                         model.add(tf.keras.layers.Dropout(taxaDr
opout))
                                         model.add(tf.keras.layers.Dense(10, acti
vation='softmax'))
                                         model.compile(optimizer='adam',
                                         loss='sparse categorical crossentropy',
                                         metrics=['accuracy'])
                                         model.fit(x_train, y_train, epochs=epoca
s)
                                         value = model.evaluate(x test, y test)
                                         model_json = model.to_json()
                                         json file = open("model CNN2.json", "w")
                                         json_file.write(model_json)
```

```
json_file.close()
                                        model.save_weights("model CNN2.h5")
                                        print("Model saved to disk")
                                        os.getcwd()
                                        somaDasEficienciasDeCadaIteracao = value
[1] + somaDasEficienciasDeCadaIteracao
                                myMutex.acquire()
                                numeroDeNeuronios.append(filtros)
                                numeroDeEpocas.append(epocas)
                                numeroDeCamadas.append(camadas)
                                numeroDeDropout.append(taxaDropout)
                                taxaDeAcertos.append(somaDasEficienciasDeCadaIte
racao/iteracaoMedia)
                                myMutex.release()
                                # Reiniciamos a soma.
                                somaDasEficienciasDeCadaIteracao = 0
# Vamos colocar uma thread para treinar cada rede com um numero especifico de ca
madas.
def thread3Camadas(camadas):
        # Para tirar a media das iteracoes, somaremos todas aqui e dividiremos p
elo total.
        somaDasEficienciasDeCadaIteracao = 0
        # Os valores que utilizaremos para dropout variarao de 10% a 90% (instru
cao abaixo).
        valoresDropout = range(10, 40, 10)# Variaremos de 10% em 10%.
        valoresDropout = [i/100 for i in valoresDropout]# Converte de porcentage
m para escala de 0 a 1.
        # Testando resultados com diferentes quantidades de epocas.
        for epocas in [2, 6]:
                # Testando resultados com diferentes quantidades de filtros.
                for filtros in [32, 64]:
                        # So para indicar em que passo da execucao estamos.
                        print("\n\nepocas: " + str(epocas) + "\nCAMADAS" + str(c
amadas) + ": " + str(filtros) + "\n\n")
                        # Este loop fica responsável por treinar com diferentes
 taxas de dropout.
                        # "i" eh o valor a cada iteracao.
                        for taxaDropout in valoresDropout:
                                # Repetimos o treinamento algumas vezes para tir
ar uma media da eficiencia
                                for iteracaoMedia in range(1,3):
                                        mnist = tf.keras.datasets.mnist
                                        (x train, y_train),(x_test, y_test) = mn
ist.load data()
                                        # reshape to be [samples][width][height]
[pixels]
                                        x_train = x_train.reshape(x_train.shape[
```

```
0], 28, 28, 1)
                                        x test = x test.reshape(x test.shape[0],
28, 28, 1)
                                        x train, x test = x train / 255.0, x tes
t / 255.0
                                        model = tf.keras.models.Sequential()
                                        model.add(tf.keras.layers.Conv2D(filtros
, kernel size=(3, 3),
                                          activation='relu',
                                         input shape=(28, 28, 1))
                                        model.add(tf.keras.layers.Conv2D(filtros
*2, (2, 2), activation='relu'))
                                        model.add(tf.keras.layers.MaxPooling2D(p
ool size=(2, 2)))
                                        model.add(tf.keras.layers.Dropout(taxaDr
opout))
                                        model.add(tf.keras.layers.Flatten())
                                        model.add(tf.keras.layers.Dense(128, act
ivation='relu'))
                                        model.add(tf.keras.layers.Dropout(taxaDr
opout))
                                        model.add(tf.keras.layers.Dense(10, acti
vation='softmax'))
                                        model.compile(optimizer='adam',
                                         loss='sparse categorical crossentropy',
                                          metrics=['accuracy'])
                                         model.fit(x train, y train, epochs=epoca
s)
                                         value = model.evaluate(x test, y test)
                                        model json = model.to json()
                                         json_file = open("model_CNN3.json", "w")
                                         json file.write(model json)
                                         json file.close()
                                        model.save weights("model CNN3.h5")
                                         print("Model saved to disk")
                                         os.getcwd()
                                         somaDasEficienciasDeCadaIteracao = value
[1] + somaDasEficienciasDeCadaIteracao
                                myMutex.acquire()
                                numeroDeNeuronios.append(filtros)
                                numeroDeEpocas.append(epocas)
                                numeroDeCamadas.append(camadas)
                                numeroDeDropout.append(taxaDropout)
                                taxaDeAcertos.append(somaDasEficienciasDeCadaIte
racao/iteracaoMedia)
                                myMutex.release()
                                # Reiniciamos a soma.
                                somaDasEficienciasDeCadaIteracao = 0
# Vamos colocar uma thread para treinar cada rede com um numero especifico de ca
madas.
def thread4Camadas(camadas):
        # Para tirar a media das iteracoes, somaremos todas aqui e dividiremos p
elo total.
        somaDasEficienciasDeCadaIteracao = 0
```

```
# Os valores que utilizaremos para dropout variarao de 10% a 90% (instru
cao abaixo).
        valoresDropout = range(10, 40, 10)# Variaremos de 10% em 10%.
        valoresDropout = [i/100 for i in valoresDropout]# Converte de porcentage
m para escala de 0 a 1.
        # Testando resultados com diferentes quantidades de epocas.
        for epocas in [2, 6]:
                # Testando resultados com diferentes quantidades de filtros.
                for filtros in [32, 64]:
                        # So para indicar em que passo da execucao estamos.
                        print("\n\nepocas: " + str(epocas) + "\nCAMADAS" + str(c
amadas) + ": " + str(filtros) + "\n\n")
                        # Este loop fica responsável por treinar com diferentes
 taxas de dropout.
                        # "i" eh o valor a cada iteracao.
                        for taxaDropout in valoresDropout:
                                # Repetimos o treinamento algumas vezes para tir
ar uma media da eficiencia
                                for iteracaoMedia in range(1,3):
                                        mnist = tf.keras.datasets.mnist
                                         (x train, y train),(x test, y test) = mn
ist.load data()
                                        # reshape to be [samples][width][height]
[pixels]
                                        x train = x train.reshape(x train.shape[
0], 28, 28, 1)
                                        x test = x test.reshape(x test.shape[0],
28, 28, 1)
                                        x train, x test = x train / 255.0, x tes
t / 255.0
                                        model = tf.keras.models.Sequential()
                                        model.add(tf.keras.layers.Conv2D(filtros
, kernel size=(3, 3),
                                         activation='relu',
                                        input shape=(28, 28, 1))
                                        model.add(tf.keras.layers.Conv2D(filtros
*2, (3, 3), activation='relu'))
                                        model.add(tf.keras.layers.MaxPooling2D(p
ool size=(3, 3)))
                                        model.add(tf.keras.layers.Conv2D(filtros
*2, (3, 3), activation='relu'))
                                        model.add(tf.keras.layers.MaxPooling2D(p
ool_size=(3, 3)))
                                        model.add(tf.keras.layers.Dropout(taxaDr
opout))
                                        model.add(tf.keras.layers.Flatten())
                                        model.add(tf.keras.layers.Dense(128, act
ivation='relu'))
                                        model.add(tf.keras.layers.Dropout(taxaDr
opout))
                                        model.add(tf.keras.layers.Dense(10, acti
vation='softmax'))
                                        model.compile(optimizer='adam',
                                         loss='sparse categorical crossentropy',
                                         metrics=['accuracy'])
```

```
model.fit(x_train, y_train, epochs=epoca
s)
                                         value = model.evaluate(x test, y test)
                                         model_json = model.to_json()
                                         json file = open("model CNN4.json", "w")
                                         json_file.write(model_json)
                                         json_file.close()
                                         model.save weights("model CNN4.h5")
                                         print("Model saved to disk")
                                         os.getcwd()
                                         somaDasEficienciasDeCadaIteracao = value
[1] + somaDasEficienciasDeCadaIteracao
                                 myMutex.acquire()
                                 numeroDeNeuronios.append(filtros)
                                 numeroDeEpocas.append(epocas)
                                 numeroDeCamadas.append(camadas)
                                 numeroDeDropout.append(taxaDropout)
                                 taxaDeAcertos.append(somaDasEficienciasDeCadaIte
racao/iteracaoMedia)
                                 myMutex.release()
                                 # Reiniciamos a soma.
                                 somaDasEficienciasDeCadaIteracao = 0
if __name__ == '__main__':
        camadas1 = threading.Thread(target=thread1Camadas,args=(1,))
        camadas2 = threading.Thread(target=thread2Camadas,args=(2,))
        camadas3 = threading.Thread(target=thread3Camadas,args=(3,))
        camadas4 = threading.Thread(target=thread4Camadas,args=(4,))
        camadas1.start()
        camadas2.start()
        camadas3.start()
        camadas4.start()
        try:
                camadas4.join();
        except:
                pass;
        try:
                camadas3.join();
        except:
                pass;
        try:
                camadas2.join();
        except:
                pass;
        try:
                camadas1.join();
```

```
except:
    pass;

listasFile = open("listasCONV.txt", "w")
listasFile.write(str(numeroDeNeuronios) + "\n")
listasFile.write(str(numeroDeEpocas) + "\n")
listasFile.write(str(numeroDeCamadas) + "\n")
listasFile.write(str(numeroDeDropout) + "\n")
listasFile.write(str(taxaDeAcertos) + "\n")
listasFile.close()
```

```
epocas: 2
CAMADAS1: 32
epocas: 2
CAMADAS2: 32
epocas: 2
CAMADAS3: 32
epocas: 2
CAMADAS4: 32
Epoch 1/2
 288/60000 [.....] - ETA: 1:33 - loss:
1.8135 - acc: 0.4167Epoch 1/2
 992/60000 [.....] - ETA: 1:21 - loss:
2.0883 - acc: 0.2984Epoch 1/
1504/60000 [.....] - ETA: 1:04 - loss:
1.7619 - acc: 0.4116Epoch 1/2
60000/60000 [============ ] - 49s 822us/sample -
loss: 0.1270 - acc: 0.9617
Epoch 2/2
60000/60000 [=========== ] - 52s 870us/sample -
loss: 0.1650 - acc: 0.9486
Epoch 2/2
loss: 0.1242 - acc: 0.9617
Epoch 2/2
60000/60000 [=========== ] - 52s 866us/sample -
loss: 0.1229 - acc: 0.9619
 768/60000 [.....] - ETA: 48s - loss: 0.
0613 - acc: 0.9831Epoch 2/2
60000/60000 [=========== ] - 51s 851us/sample -
loss: 0.0459 - acc: 0.9858
60000/60000 [==============] - 52s 859us/sample -
loss: 0.0510 - acc: 0.9843
60000/60000 [============= ] - 51s 847us/sample -
loss: 0.0430 - acc: 0.9866
60000/60000 [=========== ] - 51s 856us/sample -
loss: 0.0432 - acc: 0.9869
10000/10000 [============= ] - 6s 559us/sample - l
oss: 0.0365 - acc: 0.9876
1376/10000 [===>.....] 3904/10000 [=======
=>.....] - ETA: 4s - loss: 0.0401 - acc: 0.9855 - ET
A: 2s - loss: 0.0512 - acc: 0.9834Model saved to disk
oss: 0.0422 - acc: 0.9862
```

```
oss: 0.0334 - acc: 0.9891
334 - acc: 0.9892Model saved to disk
328 - acc: 0.989
Model saved to disk
10000/10000 [============] - 3s 327us/sample - l
oss: 0.0323 - acc: 0.9896
Model saved to disk
Epoch 1/2
9504/60000 [===>.....] - ETA: 20s - loss: 0.
3375 - acc: 0.8932Epoch 1/2
10592/60000 [====>...... - ETA: 19s - loss: 0.
3201 - acc: 0.8991Epoch 1/2
11136/60000 [====>.....] - ETA: 19s - loss: 0.
3094 - acc: 0.9027Epoch 1/2
60000/60000 [============ ] - 46s 769us/sample -
loss: 0.1238 - acc: 0.9611
1915 - acc: 0.9385Epoch 2/2
60000/60000 [=========== ] - 52s 872us/sample -
loss: 0.1215 - acc: 0.9631
Epoch 2/2
60000/60000 [=========== ] - 53s 876us/sample -
loss: 0.1654 - acc: 0.9469
 768/60000 [.....] - ETA: 51s - loss: 0.
0354 - acc: 0.9909Epoch 2/2
1856/60000 [.....]60000/60000 [========
9614
- ETA: 50s - loss: 0.0392 - acc: 0.9898 - ETA: 38s - loss: 0.0414
- acc: 0.9876Epoch 2/2
44512/60000 [============>.....] - ETA: 13s - loss: 0.
0422 - acc: 0.987660000/60000 [============= ] - 5
Os 839us/sample - loss: 0.0434 - acc: 0.9865
10000/10000 [============= ] - 7s 686us/sample - l
oss: 0.0344 - acc: 0.9885
418 - acc: 0.9875Model saved to disk
60000/60000 [=========== ] - 50s 840us/sample -
loss: 0.0427 - acc: 0.9863
loss: 0.0496 - acc: 0.9850
417 - acc: 0.9875Epoch 1/
loss: 0.0416 - acc: 0.9875
10000/10000 [============= ] 5696/60000 [=
>.....] - 4s 435us/sample - loss: 0.0402 -
acc: 0.9862
6464/10000 [=========>.....] - ETA: 1s - loss: 0.0
276 - acc: 0.990
Model saved to disk
10000/10000 [============] - 5s 499us/sample - l
oss: 0.0395 - acc: 0.9864
8480/60000 [===>.....] - ETA: 32s - loss: 0.
4509 - acc: 0.8598Model saved to dis
```

```
10000/10000 [===========] 8864/60000 [===
>.....] - 5s 490us/sample - loss: 0.0226 - ac
c: 0.9920
10176/60000 [====>...... - ETA: 30s - loss: 0.
4048 - acc: 0.8755Model saved to disk
16736/60000 [======>.....] - ETA: 22s - loss: 0.
3068 - acc: 0.9071Epoch 1/2
3808/60000 [>.....] - ETA: 41s - loss: 0.
6087 - acc: 0.8072Epoch 1/2
23264/60000 [======>.....] - ETA: 19s - loss: 0.
2558 - acc: 0.9233Epoch 1/2
loss: 0.1517 - acc: 0.9544
42400/60000 [===========>.....] - ETA: 14s - loss: 0.
1568 - acc: 0.9520Epoch 2/2
60000/60000 [=======]17280/60000 [======
>.....] - 50s 831us/sample - loss: 0.1301 - acc:
0.9600
- ETA: 37s - loss: 0.0582 - acc: 0.9823Epoch 2/2
loss: 0.1961 - acc: 0.9370
24256/60000 [=======>.....] - ETA: 30s - loss: 0.
0578 - acc: 0.9824
loss: 0.1372 - acc: 0.9575
1120/60000 [...... - ETA: 53s - loss: 0.
0643 - acc: 0.9812Epoch 2/2
60000/60000 [=========== ] - 51s 858us/sample -
loss: 0.0561 - acc: 0.9828
10000/10000 [============ ] - 7s 663us/sample - l
oss: 0.0444 - acc: 0.9855
44224/60000 [==============>.....] - ETA: 13s - loss: 0.
0500 - acc: 0.9847Model saved to disk
loss: 0.0476 - acc: 0.9852
60000/60000 [============= ] - 49s 812us/sample -
loss: 0.0613 - acc: 0.9812
10000/10000 [============= ] - 7s 687us/sample - l
oss: 0.0332 - acc: 0.9880
loss: 0.0504 - acc: 0.9844
- ETA: 36s - loss: 0.3671 - acc: 0.8889 2688/10000 [======
>.....] - ETA: 4s - loss: 0.0465 - acc: 0.9855Mod
el saved to disk
10000/10000 [==============116960/60000 [=======
>.....] - 5s 456us/sample - loss: 0.0287 - acc:
0.9910
18208/60000 [======>....] - ETA: 27s - loss: 0.
2861 - acc: 0.9138Model saved to disk
253 - acc: 0.9915
10000/10000 [============= ] - 4s 429us/sample - l
oss: 0.0251 - acc: 0.9916
 992/60000 [.....] - ETA: 54s - loss: 1.
2646 - acc: 0.5978Model saved to disk
8960/60000 [===>.....] - ETA: 30s - loss: 0.
3878 - acc: 0.8802Epoch 1/2
```

```
10528/60000 [====>.....]30944/60000 [=======
====>.....] - ETA: 28s - loss: 0.3538 - acc: 0.8903 - E
TA: 17s - loss: 0.2135 - acc: 0.9359Epoch 1/2
loss: 0.1497 - acc: 0.9546
Epoch 2/2
loss: 0.1308 - acc: 0.9597
22080/60000 [======>.....] - ETA: 31s - loss: 0.
0559 - acc: 0.9824Epoch 2/2
60000/60000 [============ ] - 52s 870us/sample -
loss: 0.1849 - acc: 0.9408
432 - acc: 0.9552Epoch 2/2
60000/60000 [=========== ] - 52s 861us/sample -
loss: 0.1431 - acc: 0.9552
 384/60000 [.....] - ETA: 43s - loss: 0.
0399 - acc: 0.9870Epoch 2/2
25632/60000 [=======>.....]60000/60000 [=======
Os 833us/sample - loss: 0.0557 - acc: 0.9826
10000/10000 [============= ] - 7s 692us/sample - l
oss: 0.0404 - acc: 0.9863
TA: 20s - loss: 0.0542 - acc: 0.9831Model saved to disk
40320/60000 [==========>.....] - ETA: 16s - loss: 0.
0610 - acc: 0.9809Epoch 1/2
60000/60000 [=========== ] - 51s 849us/sample -
loss: 0.0487 - acc: 0.9850
oss: 0.0326 - acc: 0.9892
18240/60000 [======>....] - ETA: 34s - loss: 0.
2834 - acc: 0.9126Model saved to disk
loss: 0.0509 - acc: 0.9845
8288/10000 [============>.....] - ETA: 0s - loss: 0.0
270 - acc: 0.9906Epoch 1/2
10000/10000 [=============] - 5s 457us/sample - l
oss: 0.0252 - acc: 0.9914
10000/10000 [=============] - 5s 486us/sample - l
oss: 0.0356 - acc: 0.9888
29696/60000 [======>......]
31072/60000 [==========>.....] - ETA: 20s - loss: 0.
2140 - acc: 0.9350Model saved to disk
41152/60000 [==========>.....] - ETA: 12s - loss: 0.
1846 - acc: 0.9438Epoch 1/2
 416/60000 [.....] - ETA: 2:32 - loss:
2.2737 - acc: 0.1514
60000/60000 [============ ] - 43s 712us/sample -
loss: 0.1542 - acc: 0.9530
30432/60000 [========>.....] - ETA: 22s - loss: 0.
2103 - acc: 0.9360Epoch 2/2
60000/60000 [=========== ] - 49s 809us/sample -
loss: 0.1484 - acc: 0.9548
2581 - acc: 0.9160
60000/60000 [============] - 53s 890us/sample -
loss: 0.2196 - acc: 0.9287
```

```
Epoch 2/2
loss: 0.1605 - acc: 0.9515
Epoch 2/2
60000/60000 [============ ] - 50s 834us/sample -
loss: 0.0629 - acc: 0.9803
10000/10000 [=============] - 7s 700us/sample - l
oss: 0.0408 - acc: 0.9868
39520/60000 [=========>.....]Model saved to disk -
ETA: 16s - loss: 0.0585 - acc: 0.9827
28416/60000 [========>....] - ETA: 26s - loss: 0.
0802 - acc: 0.9753Epoch 1/2
60000/60000 [============ ] - 50s 827us/sample -
loss: 0.0571 - acc: 0.9830
10000/10000 [============= ] - 7s 707us/sample - l
oss: 0.0322 - acc: 0.9888
749 - acc: 0.9769
599 - acc: 0.9823Epoch 1/2
60000/60000 [============ ] - 50s 826us/sample -
loss: 0.0742 - acc: 0.9773
60000/60000 [============ ] - 49s 815us/sample -
oss: 0.0306 - acc: 0.9902
10000/10000 [============] - 6s 606us/sample - l
oss: 0.0297 - acc: 0.9896
11744/60000 [====>......]44608/60000 [=======
======>.....] - ETA: 33s - loss: 0.3251 - acc: 0.8971 - E
TA: 11s - loss: 0.1879 - acc: 0.9429Model saved to disk
12544/60000 [=====>.....] - ETA: 32s - loss: 0.
3172 - acc: 0.8996Model saved to disk
22912/60000 [=======>.....] - ETA: 23s - loss: 0.
loss: 0.1613 - acc: 0.9513
Epoch 2/2
34720/60000 [=========>.....] - 46s 767us/sample -
loss: 0.1442 - acc: 0.9555
- ETA: 22s - loss: 0.3195 - acc: 0.8963Epoch 2/2
60000/60000 [=========== ] - 53s 891us/sample -
loss: 0.2312 - acc: 0.9258
Epoch 2/2
loss: 0.0632 - acc: 0.9804
60000/60000 [=========== ] - 54s 895us/sample -
loss: 0.1590 - acc: 0.9503
Epoch 2/2
oss: 0.0400 - acc: 0.9870
10368/60000 [====>.....] - ETA: 39s - loss: 0.
0609 - acc: 0.9797Model saved to disk
epocas: 2
CAMADAS3: 64
```

```
17536/60000 [======>.....] - ETA: 32s - loss: 0.
0863 - acc: 0.9731Epoch 1/2
60000/60000 [============ ] - 52s 859us/sample -
loss: 0.0584 - acc: 0.9825
10000/10000 [============= ] - 8s 801us/sample - l
oss: 0.0332 - acc: 0.9883
43712/60000 [==============>.....] - ETA: 14s - loss: 0.
0614 - acc: 0.981
Model saved to disk
44832/60000 [===========>......]
epocas: 2
CAMADAS1: 64
26208/60000 [========>....] - ETA: 13s - loss: 0.
0811 - acc: 0.9744
49568/60000 [==============>.....] - ETA: 9s - loss: 0.0
608 - acc: 0.9817Epoch 1/2
60000/60000 [============ ] - 53s 877us/sample -
loss: 0.0794 - acc: 0.9753
60000/60000 [=========== ] - 54s 892us/sample -
oss: 0.0337 - acc: 0.9896
340 - acc: 0.9887Model saved to disk
epocas: 2
CAMADAS4: 64
oss: 0.0337 - acc: 0.9888
227 - acc: 0.9617Model saved to disk
20256/60000 [======>.....] - ETA: 39s - loss: 0.
1842 - acc: 0.9417
epocas: 2
CAMADAS2: 64
27136/60000 [========>.....]60000/60000 [=======
  5s 917us/sample - loss: 0.1148 - acc: 0.9643
Epoch 2/2
1024/60000 [.....] - ETA: 41s - loss: 0.
0322 - acc: 0.9922Epoch 1/2
29760/60000 [========>.....] - ETA: 27s - loss: 0.
1474 - acc: 0.9535Epoch 1/2
60000/60000 [============= ] - 69s 1ms/sample - lo
ss: 0.1064 - acc: 0.9671
30304/60000 [=======>:....]33920/60000 [======
=====>.....] - ETA: 42s - loss: 0.2047 - acc: 0.9343 - E
TA: 33s - loss: 0.0416 - acc: 0.9872Epoch 2/2
60000/60000 [=============== ] - 80s 1ms/sample - lo
ss: 0.0412 - acc: 0.9872
60000/60000 [=============== ] - 85s 1ms/sample - lo
```

```
ss: 0.1353 - acc: 0.9567
4352/10000 [=======>.....] - ETA: 6s - loss: 0.0
638 - acc: 0.9789Epoch 2/2
60000/60000 [============= ] - 85s 1ms/sample - lo
ss: 0.1124 - acc: 0.9646
31616/60000 [========>.....]
ss: 0.0398 - acc: 0.9861
5760/60000 [=>.....] - ETA: 1:06 - loss:
0.0546 - acc: 0.9828Model saved to disk
10272/60000 [====>.....] - ETA: 57s - loss: 0.
0485 - acc: 0.9849Epoch 1/2
60000/60000 [============== ] - 81s 1ms/sample - lo
ss: 0.0393 - acc: 0.9875
ss: 0.0373 - acc: 0.9891
Model saved to disk37952/60000 [==========>.....]
34912/60000 [==========>.....] - ETA: 32s - loss: 0.
1598 - acc: 0.9506Epoch 1/2
60000/60000 [=============== ] - 79s 1ms/sample - lo
ss: 0.0448 - acc: 0.9862
60000/60000 [============== ] - 80s 1ms/sample - lo
ss: 0.0415 - acc: 0.9869
60000/60000 [============ ] - 75s 1ms/sample - lo
ss: 0.1205 - acc: 0.9623
24320/60000 [=======>.....] - ETA: 44s - loss: 0.
1738 - acc: 0.9461Epoch 2/2
oss: 0.0270 - acc: 0.9911
26368/60000 [========>.....] - ETA: 41s - loss: 0.
1664 - acc: 0.9483Model saved to disk
oss: 0.0364 - acc: 0.9884
30080/60000 [=======>.....] 5984/60000 [=
>.....] - ETA: 35s - loss: 0.1552 - acc: 0.
9515 - ETA: 45s - loss: 0.0450 - acc: 0.988
Model saved to dis
10688/60000 [====>.....] - ETA: 39s - loss: 0.
0437 - acc: 0.9864Epoch 1/2
37920/60000 [==========>.....] - ETA: 24s - loss: 0.
1374 - acc: 0.9573Epoch 1/2
60000/60000 [============= ] - 73s 1ms/sample - lo
ss: 0.1104 - acc: 0.9661
Epoch 2/2
60000/60000 [============ ] - 73s 1ms/sample - lo
ss: 0.0420 - acc: 0.9869
ss: 0.0402 - acc: 0.9868
=====>.....] - ETA: 6s - loss: 0.1401 - acc: 0.9562 - ET
A: 35s - loss: 0.0365 - acc: 0.9887Model saved to disk
119 - acc: 0.9652Epoch 1/2
60000/60000 [=============== ] - 83s 1ms/sample - lo
ss: 0.1340 - acc: 0.9581
Epoch 2/2
60000/60000 [=============== ] - 84s 1ms/sample - lo
ss: 0.1090 - acc: 0.9659
```

In [0]:

```
# q4Final.py
import tensorflow as tf
import os
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
# reshape to be [samples][width][height][pixels]
x_{train} = x_{train.reshape}(x_{train.shape}[0], 28, 28, 1)
x_{test} = x_{test.reshape}(x_{test.shape}[0], 28, 28, 1)
x_{train}, x_{test} = x_{train} / 255.0, x_{test} / 255.0
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, kernel size=(3, 3),
 activation='relu',
input shape=(28, 28, 1))
model.add(tf.keras.layers.Conv2D(512, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Conv2D(512, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Conv2D(512, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
model.compile(optimizer='adam',
loss='sparse categorical crossentropy',
metrics=['accuracy'])
model.fit(x train, y train, epochs=10)
model.evaluate(x_test, y_test)
model json = model.to json()
json file = open("model CNN.json", "w")
json file.write(model json)
json file.close()
model.save weights("model CNN.h5")
print("Model saved to disk")
os.getcwd()
```

```
Epoch 1/10
s: 0.2189 - acc: 0.9315
Epoch 2/10
s: 0.0785 - acc: 0.9787
Epoch 3/10
s: 0.0625 - acc: 0.9830
Epoch 4/10
60000/60000 [============= ] - 78s 1ms/sample - los
s: 0.0532 - acc: 0.9855
Epoch 5/10
60000/60000 [============= ] - 78s 1ms/sample - los
s: 0.0457 - acc: 0.9871
Epoch 6/10
s: 0.0438 - acc: 0.9879
Epoch 7/10
60000/60000 [============= ] - 78s 1ms/sample - los
s: 0.0376 - acc: 0.9899
Epoch 8/10
60000/60000 [============ ] - 78s 1ms/sample - los
s: 0.0353 - acc: 0.9904
Epoch 9/10
60000/60000 [============= ] - 78s 1ms/sample - los
s: 0.0340 - acc: 0.9907
Epoch 10/10
s: 0.0335 - acc: 0.9909
10000/10000 [============ ] - 4s 428us/sample - los
s: 0.0298 - acc: 0.9935
Model saved to disk
```

Out[0]:

'/content'