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
60000/60000 [==============================] - 6s 101us/sample - los
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]:

```
'/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
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(10, activation=tf
```

```
.nn.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_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)
                                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, 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(10, activation=tf
.nn.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_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)
                                        ])
                                        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_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(neuronios, activa
tion=tf.nn.relu),
                                              tf.keras.layers.Dropout(taxaDropout),#
 Diferentes valores de dropout.
                                              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=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 [=========
=====================] - ETA: 23s - loss: 0.1076 - acc: 0.9645 - 2
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
60000/60000 [==============================] - 26s 435us/sample -
loss: 0.1005 - acc: 0.9692
Epoch 3/8
60000/60000 [==============================] - 28s 463us/sample -
loss: 0.1098 - acc: 0.9671
Epoch 3/854080/60000 [==========================>...]
60000/60000 [==============================] - 29s 485us/sample -
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/
8✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

60000/60000 [==============================] - 26s 439us/sample -
loss: 0.0723 - acc: 0.9771
Epoch 4/
8✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

60000/60000 [==============================] - 28s 465us/sample -
loss: 0.0835 - acc: 0.9741
33120/60000 [===============>..............] - ETA: 11s - loss: 0.
0527 - acc: 0.9832Epoch 4/8
60000/60000 [==============================] - 29s 481us/sample -
loss: 0.0930 - acc: 0.9727
23776/60000 [==========>...................] - ETA: 16s - loss: 0.
0547 - acc: 0.9823Epoch 4/8
60000/60000 [==============================] - 25s 415us/sample -
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
60000/60000 [==============================] - 28s 459us/sample -
loss: 0.0678 - acc: 0.9792
18240/60000 [========>.....................] - ETA: 18s - loss: 0.
0460 - acc: 0.9859Epoch 5/8
11264/60000 [====>.........................]60000/60000 [=========
=====================] - ETA: 22s - loss: 0.0532 - acc: 0.9837 - 2
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
60000/60000 [==============================] - 27s 446us/sample -
loss: 0.0512 - acc: 0.9842
Epoch 6/8
60000/60000 [==============================] - 28s 460us/sample -
loss: 0.0594 - acc: 0.9818
Epoch 6/8
60000/60000 [==============================] - 29s 484us/sample -
loss: 0.0688 - acc: 0.9795
14272/60000 [======>.......................] - ETA: 21s - loss: 0.
0422 - acc: 0.987
0✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓
Epoch 6/8
60000/60000 [==============================] - 26s 428us/sample -
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 [=========
=====================] - ETA: 9s - loss: 0.0525 - acc: 0.9843 - 28
s 458us/sample - loss: 0.0460 - acc: 0.9851
55168/60000 [==========================>...] - ETA: 2s - loss: 0.0
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
60000/60000 [==============================] - 29s 475us/sample -
loss: 0.0531 - acc: 0.9845
Epoch 8/
8✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

60000/60000 [==============================] - 26s 428us/sample -
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
10000/10000 [==============================]20096/60000 [=========
>....................] - 4s 354us/sample - loss: 0.1359 - acc: 0.9
697
47840/60000 [======================>.......] - ETA: 5s - loss: 0.0
466 - acc: 0.9859Model saved to disk
60000/60000 [==============================] - 24s 403us/sample -
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
10000/10000 [==============================] - 4s 384us/sample - l
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
60000/60000 [==============================] - 25s 416us/sample -
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.
1156 - acc: 0.9645Epoch 3/8
60000/60000 [==============================] - 25s 414us/sample -
loss: 0.0708 - acc: 0.9781
Epoch 4/8
60000/60000 [==============================] - 28s 463us/sample -
loss: 0.1139 - acc: 0.9657
33440/60000 [===============>..............] - ETA: 12s - loss: 0.
1237 - acc: 0.9644Epoch 3/
8✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

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
60000/60000 [==============================] - 25s 418us/sample -
loss: 0.0543 - acc: 0.9830
 8032/60000 [===>..........................] - ETA: 24s - loss: 0.
0825 - acc: 0.9747Epoch 5/8
60000/60000 [==============================] - 28s 459us/sample -
loss: 0.0853 - acc: 0.9740
25888/60000 [===========>..................] - ETA: 14s - loss: 0.
0406 - acc: 0.9876Epoch 4/8
60000/60000 [==============================] - 27s 443us/sample -
loss: 0.0588 - acc: 0.9809
48448/60000 [=======================>......] - ETA: 4s - loss: 0.0
409 - acc: 0.9872Epoch 5/8
60000/60000 [==============================] - 29s 483us/sample -
loss: 0.0938 - acc: 0.9725
59872/60000 [============================>.] - ETA: 0s - loss: 0.0
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
60000/60000 [==============================] - 28s 464us/sample -
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
60000/60000 [==============================] - 25s 412us/sample -
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
60000/60000 [==============================] - 26s 433us/sample -
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 -
loss: 0.0511 - acc: 0.9841
48320/60000 [=======================>......] - ETA: 4s - loss: 0.0
232 - acc: 0.9922Epoch 7/8
60000/60000 [==============================] - 25s 420us/sample -
loss: 0.0236 - acc: 0.9921
60000/60000 [==============================] - ETA: 11s - loss: 0.
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
10000/10000 [==============================] - 4s 384us/sample - l
oss: 0.0723 - acc: 0.9808
19488/60000 [========>.....................] - ETA: 19s - loss: 0.
0432 - acc: 0.9868Model saved to disk
60000/60000 [==============================]24224/60000 [=========
==>.................] - 25s 413us/sample - loss: 0.2396 - acc: 0.
9302
 - ETA: 16s - loss: 0.0442 - acc: 0.9870Epoch 2/8
48096/60000 [======================>......] - ETA: 5s - loss: 0.0
543 - acc: 0.9839Epoch 1/8
60000/60000 [==============================] - 29s 484us/sample -
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
52416/60000 [=========================>....]37376/60000 [=========
========>...........] - ETA: 3s - loss: 0.2329 - acc: 0.9288 - ET
A: 10s - loss: 0.0471 - acc: 0.9868Epoch 3/8
```

```
60000/60000 [==============================] - 27s 447us/sample -
loss: 0.2203 - acc: 0.9327
44352/60000 [=====================>........] 7808/60000 [==
>...........................]
60000/60000 [==============================] - 29s 478us/sample -
loss: 0.0504 - acc: 0.9856
10000/10000 [==============================] - 4s 387us/sample - l
oss: 0.0945 - acc: 0.9764
33888/60000 [===============>..............]37728/60000 [========
========>...........] - ETA: 12s - loss: 0.2900 - acc: 0.9103 - E
TA: 9s - loss: 0.0751 - acc: 0.9768Model saved to disk
51136/60000 [========================>.....] - ETA: 3s - loss: 0.0
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
4✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓
Epoch 4/8
60000/60000 [==============================] - 27s 448us/sample -
loss: 0.2302 - acc: 0.9297
Epoch 2/8
60000/60000 [==============================] - 25s 418us/sample -
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
58048/60000 [============================>.] - ETA: 0s - loss: 0.0
723 - acc: 0.9771Epoch 3/8
60000/60000 [==============================] - 27s 456us/sample -
loss: 0.0727 - acc: 0.9768
 1952/60000 [..............................] - ETA: 27s - loss: 0.
0908 - acc: 0.9672Epoch 4/8
60000/60000 [==============================] - 25s 425us/sample -
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 [========
======================] - ETA: 26s - loss: 0.0832 - acc: 0.9747 - 2
9s 477us/sample - loss: 0.0835 - acc: 0.9745
Epoch 4/8
60000/60000 [==============================] - 26s 431us/sample -
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
60000/60000 [==============================] - 26s 429us/sample -
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
60000/60000 [==============================] - 29s 487us/sample -
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
60000/60000 [==============================] - 26s 434us/sample -
loss: 0.0346 - acc: 0.9884
50752/60000 [========================>.....] - ETA: 4s - loss: 0.0
686 - acc: 0.9797Epoch 8/8
60000/60000 [==============================] - 28s 459us/sample -
loss: 0.0509 - acc: 0.9839
58688/60000 [============================>.] - ETA: 0s - loss: 0.0
693 - acc: 0.9797Epoch 7/8
60000/60000 [==============================] - 29s 486us/sample -
loss: 0.0694 - acc: 0.9797
 9984/60000 [===>..........................] - ETA: 23s - loss: 0.
0244 - acc: 0.9918Epoch 6/8
60000/60000 [==============================] - 26s 426us/sample -
loss: 0.2490 - acc: 0.9269
Epoch 2/8
60000/60000 [==============================] - 28s 460us/sample -
loss: 0.0333 - acc: 0.9890
60000/60000 [==============================] - 28s 459us/sample -
loss: 0.0455 - acc: 0.9859
Epoch 8/8
10000/10000 [==============================] - 4s 391us/sample - l
oss: 0.0731 - acc: 0.9803
56704/60000 [===========================>..]Model saved to disk -
ETA: 1s - loss: 0.0589 - acc: 0.9826
60000/60000 [==============================] - 29s 481us/sample -
loss: 0.0593 - acc: 0.9825
Epoch 7/
8✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

 4864/60000 [=>............................] - ETA: 24s - loss: 0.
0421 - acc: 0.9877Epoch 1/8
60000/60000 [==============================] - 24s 404us/sample -
```

```
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
10000/10000 [==============================] - 4s 363us/sample - l
oss: 0.0785 - acc: 0.9799
60000/60000 [==============================] - 28s 468us/sample -
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
44096/60000 [=====================>........] - ETA: 6s - loss: 0.1
359 - acc: 0.9593Epoch 4/8
60000/60000 [==============================] 6624/60000 [==
>...........................] - 24s 403us/sample - loss: 0.0542 -
acc: 0.9826
 - ETA: 25s - loss: 0.0616 - acc: 0.9802Epoch 6/8
60000/60000 [==============================] - 27s 449us/sample -
loss: 0.1332 - acc: 0.9600
10432/60000 [====>.........................] - ETA: 21s - loss: 0.
0445 - acc: 0.9868Epoch 3/8
17920/60000 [=======>......................]60000/60000 [========
=====================] - ETA: 19s - loss: 0.1048 - acc: 0.9670 - 3
0s 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
60000/60000 [==============================] - 29s 477us/sample -
loss: 0.1479 - acc: 0.9578
Epoch 3/
8✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

18176/60000 [========>.....................]60000/60000 [=========
=====================] - ETA: 20s - loss: 0.1150 - acc: 0.9663 - 2
7s 447us/sample - loss: 0.0637 - acc: 0.9798
Epoch 6/
8✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

60000/60000 [==============================] - 25s 422us/sample -
loss: 0.0388 - acc: 0.9875
Epoch 8/8
51328/60000 [========================>.....] - ETA: 3s - loss: 0.0
882 - acc: 0.9736Buffered data was truncated after reaching the ou
tput size limit.
```

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
60000/60000 [==============================] - 5s 84us/sample - los
s: 0.2446 - acc: 0.9246
Epoch 2/8
60000/60000 [==============================] - 5s 83us/sample - los
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
60000/60000 [==============================] - 5s 82us/sample - los
s: 0.0712 - acc: 0.9787
Epoch 7/8
60000/60000 [==============================] - 5s 83us/sample - los
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.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(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 [==============================] - 16s 268us/sample - lo
ss: 0.2004 - acc: 0.9403
Epoch 2/5
60000/60000 [==============================] - 15s 256us/sample - lo
ss: 0.0851 - acc: 0.9741
Epoch 3/5
60000/60000 [==============================] - 15s 251us/sample - lo
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]:

```
'/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
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=(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_tes
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/
2✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

 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
60000/60000 [==============================] - 52s 860us/sample - 
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
10000/10000 [==============================] - 4s 381us/sample - l
oss: 0.0422 - acc: 0.9862
10000/10000 [==============================] - 4s 403us/sample - l
```

```
oss: 0.0334 - acc: 0.9891
 7872/10000 [=====================>.......] - ETA: 0s - loss: 0.0
334 - acc: 0.9892Model saved to disk
 8672/10000 [========================>....] - ETA: 0s - loss: 0.0
328 - acc: 0.989
5✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓
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
47488/60000 [=====================>.......] - ETA: 11s - loss: 0.
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 [=========
=====================] - 53s 888us/sample - loss: 0.1258 - acc: 0.
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
0s 839us/sample - loss: 0.0434 - acc: 0.9865
10000/10000 [==============================] - 7s 686us/sample - l
oss: 0.0344 - acc: 0.9885
54240/60000 [==========================>...] - ETA: 4s - loss: 0.0
418 - acc: 0.9875Model saved to disk
60000/60000 [==============================] - 50s 840us/sample -
loss: 0.0427 - acc: 0.9863
60000/60000 [==============================] - 50s 839us/sample -
loss: 0.0496 - acc: 0.9850
59136/60000 [============================>.] - ETA: 0s - loss: 0.0
417 - acc: 0.9875Epoch 1/
2✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

60000/60000 [==============================] - 50s 839us/sample -
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
1✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓
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
```

```
k✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

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
60000/60000 [==============================] - 44s 727us/sample -
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
60000/60000 [==============================] - 53s 879us/sample -
loss: 0.1961 - acc: 0.9370
24256/60000 [===========>..................] - ETA: 30s - loss: 0.
0578 - acc: 0.9824
60000/60000 [==============================] - 53s 886us/sample -
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
51232/60000 [========================>.....] - ETA: 7s - loss: 0.0
617 - acc: 0.9809Epoch 1/2
60000/60000 [==============================] - 49s 811us/sample -
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
60000/60000 [==============================] - 50s 826us/sample -
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 [==============================]16960/60000 [=======
>.....................] - 5s 456us/sample - loss: 0.0287 - acc:
0.9910
18208/60000 [========>.....................] - ETA: 27s - loss: 0.
2861 - acc: 0.9138Model saved to disk
 9856/10000 [============================>.] - ETA: 0s - loss: 0.0
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
60000/60000 [==============================] - 43s 716us/sample -
loss: 0.1497 - acc: 0.9546
Epoch 2/2
60000/60000 [==============================] - 48s 803us/sample -
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
59712/60000 [============================>.] - ETA: 0s - loss: 0.1
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 [=========
=====================] - ETA: 30s - loss: 0.0634 - acc: 0.9808 - 5
0s 833us/sample - loss: 0.0557 - acc: 0.9826
10000/10000 [==============================] - 7s 692us/sample - l
oss: 0.0404 - acc: 0.9863
46592/60000 [======================>.......]35456/60000 [=========
=======>.............] - ETA: 11s - loss: 0.0487 - acc: 0.9851 - E
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
10000/10000 [==============================] - 7s 680us/sample - l
oss: 0.0326 - acc: 0.9892
18240/60000 [========>.....................] - ETA: 34s - loss: 0.
2834 - acc: 0.9126Model saved to disk
60000/60000 [==============================] - 50s 833us/sample -
loss: 0.0607 - acc: 0.9815
60000/60000 [==============================] - 50s 835us/sample -
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
45248/60000 [=====================>........] - ETA: 13s - loss: 0.
2581 - acc: 0.9160
60000/60000 [==============================] - 53s 890us/sample -
loss: 0.2196 - acc: 0.9287
```

```
Epoch 2/2
60000/60000 [==============================] - 53s 886us/sample -
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
53760/60000 [=========================>....] - ETA: 5s - loss: 0.0
749 - acc: 0.9769
58976/60000 [============================>.] - ETA: 0s - loss: 0.0
599 - acc: 0.9823Epoch 1/2
60000/60000 [==============================] - 50s 826us/sample -
loss: 0.0742 - acc: 0.9773
60000/60000 [==============================] - 49s 815us/sample -
loss: 0.0595 - acc: 0.9824
10000/10000 [==============================] - 6s 598us/sample - l
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.
2340 - acc: 0.9270Epoch 1/2
57312/60000 [===========================>..] - ETA: 1s - loss: 0.1
650 - acc: 0.9501Epoch 1/2
60000/60000 [==============================] - 43s 709us/sample -
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
60000/60000 [==============================] - 52s 860us/sample -
loss: 0.0632 - acc: 0.9804
60000/60000 [==============================] - 54s 895us/sample -
loss: 0.1590 - acc: 0.9503
Epoch 2/2
10000/10000 [==============================] - 7s 702us/sample - l
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
4✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓
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 -
loss: 0.0594 - acc: 0.9821
10000/10000 [==============================] - 8s 786us/sample - l
oss: 0.0337 - acc: 0.9896
 9888/10000 [============================>.] - ETA: 0s - loss: 0.0
340 - acc: 0.9887Model saved to disk


epocas: 2
CAMADAS4: 64


10000/10000 [==============================] - 8s 778us/sample - l
oss: 0.0337 - acc: 0.9888
53056/60000 [=========================>....] - ETA: 6s - loss: 0.1
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 [========
=====================] - ETA: 30s - loss: 0.1564 - acc: 0.9507 - 5
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 [==============>...............]
10000/10000 [==============================] - 11s 1ms/sample - lo
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
10000/10000 [==============================] - 11s 1ms/sample - lo
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
10000/10000 [==============================] - 9s 869us/sample - l
oss: 0.0270 - acc: 0.9911
26368/60000 [============>.................] - ETA: 41s - loss: 0.
1664 - acc: 0.9483Model saved to disk
10000/10000 [==============================] - 9s 881us/sample - l
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
1✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓
Model saved to dis
k✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

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
10000/10000 [==============================] - 11s 1ms/sample - lo
ss: 0.0402 - acc: 0.9868
55584/60000 [==========================>...]33376/60000 [========
======>..............] - ETA: 6s - loss: 0.1401 - acc: 0.9562 - ET
A: 35s - loss: 0.0365 - acc: 0.9887Model saved to disk
57024/60000 [===========================>..] - ETA: 4s - loss: 0.1
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
```

```
Epoch 2/2
 5856/60000 [=>............................] - ETA: 1:17 - loss:
0.0470 - acc: 0.984
1✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓
Buffered data was truncated after reaching the output size limit.
```

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
60000/60000 [==============================] - 79s 1ms/sample - los
s: 0.2189 - acc: 0.9315
Epoch 2/10
60000/60000 [==============================] - 78s 1ms/sample - los
s: 0.0785 - acc: 0.9787
Epoch 3/10
60000/60000 [==============================] - 78s 1ms/sample - los
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
60000/60000 [==============================] - 78s 1ms/sample - los
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
60000/60000 [==============================] - 78s 1ms/sample - los
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'
```