Análisis de Datos y Aprendizaje Máquina con Tensorflow 2.0: Redes neuronales convolucionales

2019/09/30

Redes Neuronales Convolucionales Profundas y Regularización

- Objetivo: Implementar redes convolucionales profundas, conocer el desempeño de los optimizadores y los efectos de regularización en el entrenamiento. Se conocerá el efecto de BatchNormalization antes y después de la activación
- Se apilan dos a tres capas convolucionales en redes VGG como muestra K. Simonyan y A. Zisserman en "Very Deep Convolutional Networks for Large-Scale Image Recognition" https://arxiv.org/abs/1409.1556

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras



fashion_mnist = keras.datasets.fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

In [2]:
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                        'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

In [3]:
```
for i in range(5):
    rand_image_idx = np.random.randint(0, y_train.shape[0])
    plt.subplot(1, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid('off')
    plt.imshow(x_train[rand_image_idx])
    plt.xlabel(class_names[y_train[rand_image_idx]])
plt.show()
```


```
In [4]: # escalar entre 0 y 1
        x_train = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32') / 255
        x_test = x_test.reshape(x_test.shape[0], 28, 28, 1).astype('float32') / 255


        print(x_train.shape) # (60000, 28, 28, 1)
        print(x_test.shape)  # (10000, 28, 28, 1)
```

```
(60000, 28, 28, 1)
(10000, 28, 28, 1)
```

Obtener dimensiones

```
In [5]: x, y, channel = x_train.shape[1:]

        input_shape = (x, y, channel)
```

```
In [6]: epoch = 20
        verbose = 1
        batch = 50
```

Deep CNN

- Red CNN profunda con 3 bloques de Conv2D y MaxPooling2D
- La activación es 'LeakyReLU'

```
In [7]: from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D, Lea
```

```
In [8]: def cnn():
            model = Sequential()

            model.add(Conv2D(20, (3,3), padding = 'same', activation=None, input_shape = input_sh
            model.add(LeakyReLU())
            model.add(Conv2D(20, (3,3), padding = 'same', activation=None))
            model.add(LeakyReLU())
            model.add(MaxPooling2D((2,2)))
```

```
    model.add(Conv2D(40, (2,2), padding = 'same', activation=None))
    model.add(LeakyReLU())
    model.add(Conv2D(40, (2,2), padding = 'same', activation=None))
    model.add(LeakyReLU())
    model.add(MaxPooling2D((2,2)))

    model.add(Conv2D(60, (2,2), padding = 'same', activation=None))
    model.add(LeakyReLU())
    model.add(Conv2D(60, (2,2), padding = 'same', activation=None))
    model.add(LeakyReLU())
    model.add(MaxPooling2D((2,2)))


    model.add(Flatten())

    model.add(Dense(32, activation = None))
    model.add(LeakyReLU())
    model.add(Dense(32, activation = None))
    model.add(LeakyReLU())
    model.add(Dense(10, activation = 'softmax'))


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

```
In [9]: model = cnn()
```

```
In [10]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 20)        200
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 28, 28, 20)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 20)        3620
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 28, 28, 20)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 20)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 40)        3240
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 14, 14, 40)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 40)        6440
_________________________________________________________________
```

```
leaky_re_lu_3 (LeakyReLU)    (None, 14, 14, 40)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 40)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 60)          9660
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 7, 7, 60)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 60)          14460
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 7, 7, 60)          0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 60)          0
_________________________________________________________________
flatten (Flatten)            (None, 540)               0
_________________________________________________________________
dense (Dense)                (None, 32)                17312
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 56,318
Trainable params: 56,318
Non-trainable params: 0
_________________________________________________________________
```

```
In [11]: history1 = model.fit(x_train, y_train, batch_size = batch, validation_split = 0.3,
                              epochs = epoch, verbose = verbose)

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 11s 261us/sample - loss: 0.5652 - accuracy: 0.7946
Epoch 2/20
42000/42000 [==============================] - 9s 208us/sample - loss: 0.3387 - accuracy: 0.8776
Epoch 3/20
42000/42000 [==============================] - 9s 213us/sample - loss: 0.2860 - accuracy: 0.8964
Epoch 4/20
42000/42000 [==============================] - 9s 215us/sample - loss: 0.2571 - accuracy: 0.9062
Epoch 5/20
42000/42000 [==============================] - 9s 211us/sample - loss: 0.2357 - accuracy: 0.9147
Epoch 6/20
42000/42000 [==============================] - 9s 211us/sample - loss: 0.2191 - accuracy: 0.9193
Epoch 7/20
42000/42000 [==============================] - 9s 212us/sample - loss: 0.2030 - accuracy: 0.9250
```

```
Epoch 8/20
42000/42000 [==============================] - 9s 212us/sample - loss: 0.1891 - accuracy: 0.9305
Epoch 9/20
42000/42000 [==============================] - 9s 215us/sample - loss: 0.1763 - accuracy: 0.9356
Epoch 10/20
42000/42000 [==============================] - 9s 209us/sample - loss: 0.1686 - accuracy: 0.9370
Epoch 11/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.1570 - accuracy: 0.9409
Epoch 12/20
42000/42000 [==============================] - 9s 210us/sample - loss: 0.1464 - accuracy: 0.9448
Epoch 13/20
42000/42000 [==============================] - 9s 205us/sample - loss: 0.1383 - accuracy: 0.9485
Epoch 14/20
42000/42000 [==============================] - 9s 211us/sample - loss: 0.1316 - accuracy: 0.9501
Epoch 15/20
42000/42000 [==============================] - 9s 216us/sample - loss: 0.1229 - accuracy: 0.9545
Epoch 16/20
42000/42000 [==============================] - 9s 211us/sample - loss: 0.1135 - accuracy: 0.9572
Epoch 17/20
42000/42000 [==============================] - 9s 212us/sample - loss: 0.1077 - accuracy: 0.9599
Epoch 18/20
42000/42000 [==============================] - 9s 214us/sample - loss: 0.1043 - accuracy: 0.9604
Epoch 19/20
42000/42000 [==============================] - 9s 212us/sample - loss: 0.0944 - accuracy: 0.9651
Epoch 20/20
42000/42000 [==============================] - 9s 214us/sample - loss: 0.0904 - accuracy: 0.9660
```

```
In [12]: test_loss, test_acc = model.evaluate(x_test, y_test, verbose = 0)

         print('\nTest acccuracy:', test_acc)
```

```
Test acccuracy: 0.9125
```

- Red CNN profunda con 2 bloques de Conv2D y MaxPooling2D

```
In [13]: def cnn():
             model = Sequential()


             model.add(Conv2D(20, (3,3), padding = 'same', activation=None, input_shape = input_s
             model.add(LeakyReLU())
             model.add(Conv2D(20, (3,3), padding = 'same', activation=None))
             model.add(LeakyReLU())
             model.add(MaxPooling2D((2,2)))

             model.add(Conv2D(40, (2,2), padding = 'same', activation=None))
             model.add(LeakyReLU())
```

```
    model.add(Conv2D(40, (2,2), padding = 'same', activation=None))
    model.add(LeakyReLU())
    model.add(MaxPooling2D((2,2)))

    model.add(Flatten())

    model.add(Dense(32, activation = None))
    model.add(LeakyReLU())
    model.add(Dense(32, activation = None))
    model.add(LeakyReLU())
    model.add(Dense(10, activation = 'softmax'))

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

```
In [14]: model = cnn()
         model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 28, 28, 20)        200
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 28, 28, 20)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 20)        3620
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 28, 28, 20)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 20)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 40)        3240
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 14, 14, 40)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 40)        6440
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 14, 14, 40)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 40)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1960)              0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                62752
_________________________________________________________________
```

```
leaky_re_lu_12 (LeakyReLU)   (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                1056
_________________________________________________________________
leaky_re_lu_13 (LeakyReLU)   (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                330
=================================================================
Total params: 77,638
Trainable params: 77,638
Non-trainable params: 0
_________________________________________________________________
```

```
In [15]: history2 = model.fit(x_train, y_train, batch_size = batch, validation_split = 0.3,
                              epochs = epoch, verbose = verbose)
```

```
Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 8s 191us/sample - loss: 0.4939 - accuracy: 0.8229
Epoch 2/20
42000/42000 [==============================] - 7s 177us/sample - loss: 0.3115 - accuracy: 0.8886
Epoch 3/20
42000/42000 [==============================] - 7s 177us/sample - loss: 0.2624 - accuracy: 0.9053
Epoch 4/20
42000/42000 [==============================] - 7s 177us/sample - loss: 0.2323 - accuracy: 0.9147
Epoch 5/20
42000/42000 [==============================] - 8s 180us/sample - loss: 0.2082 - accuracy: 0.9225
Epoch 6/20
42000/42000 [==============================] - 8s 179us/sample - loss: 0.1893 - accuracy: 0.9302
Epoch 7/20
42000/42000 [==============================] - 7s 178us/sample - loss: 0.1712 - accuracy: 0.9363
Epoch 8/20
42000/42000 [==============================] - 8s 179us/sample - loss: 0.1565 - accuracy: 0.9421
Epoch 9/20
42000/42000 [==============================] - 8s 181us/sample - loss: 0.1413 - accuracy: 0.9482
Epoch 10/20
42000/42000 [==============================] - 7s 179us/sample - loss: 0.1313 - accuracy: 0.9502
Epoch 11/20
42000/42000 [==============================] - 8s 181us/sample - loss: 0.1157 - accuracy: 0.9567
Epoch 12/20
42000/42000 [==============================] - 8s 190us/sample - loss: 0.1067 - accuracy: 0.9602
Epoch 13/20
42000/42000 [==============================] - 8s 190us/sample - loss: 0.0977 - accuracy: 0.9631
Epoch 14/20
42000/42000 [==============================] - 8s 189us/sample - loss: 0.0895 - accuracy: 0.9657
Epoch 15/20
42000/42000 [==============================] - 8s 186us/sample - loss: 0.0802 - accuracy: 0.9705
Epoch 16/20
42000/42000 [==============================] - 8s 186us/sample - loss: 0.0751 - accuracy: 0.9718
```

```
Epoch 17/20
42000/42000 [==============================] - 8s 187us/sample - loss: 0.0697 - accuracy: 0.9739
Epoch 18/20
42000/42000 [==============================] - 8s 193us/sample - loss: 0.0631 - accuracy: 0.9773
Epoch 19/20
42000/42000 [==============================] - 8s 187us/sample - loss: 0.0593 - accuracy: 0.9781
Epoch 20/20
42000/42000 [==============================] - 8s 183us/sample - loss: 0.0548 - accuracy: 0.9794
```

```
In [16]: test_loss, test_acc = model.evaluate(x_test, y_test, verbose = 0)

         print('\nTest acccuracy:', test_acc)
```

```
Test acccuracy: 0.9081
```

Regularización

- Batch norm antes de activación con RMSprop

```
In [17]: from tensorflow.keras.layers import BatchNormalization
```

```
In [18]: def cnn():
             model = Sequential()


             model.add(Conv2D(20, (3,3), padding = 'same', activation=None, input_shape = input_s
             model.add(BatchNormalization())
             model.add(LeakyReLU())
             model.add(Conv2D(20, (3,3), padding = 'same', activation=None))
             model.add(BatchNormalization())
             model.add(LeakyReLU())
             model.add(MaxPooling2D((2,2)))

             model.add(Conv2D(40, (2,2), padding = 'same', activation=None))
             model.add(BatchNormalization())
             model.add(LeakyReLU())
             model.add(Conv2D(40, (2,2), padding = 'same', activation=None))
             model.add(BatchNormalization())
             model.add(LeakyReLU())
             model.add(MaxPooling2D((2,2)))


             model.add(Flatten())

             model.add(Dense(32, activation = None))
             model.add(BatchNormalization())
             model.add(LeakyReLU())
```

```
    model.add(Dense(32, activation = None))
    model.add(BatchNormalization())
    model.add(LeakyReLU())
    model.add(Dense(10, activation = 'softmax'))

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

```
In [19]: model = cnn()
         model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 28, 28, 20)        200
_________________________________________________________________
batch_normalization (BatchNo (None, 28, 28, 20)        80
_________________________________________________________________
leaky_re_lu_14 (LeakyReLU)   (None, 28, 28, 20)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 28, 28, 20)        3620
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 20)        80
_________________________________________________________________
leaky_re_lu_15 (LeakyReLU)   (None, 28, 28, 20)        0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 20)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 40)        3240
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 40)        160
_________________________________________________________________
leaky_re_lu_16 (LeakyReLU)   (None, 14, 14, 40)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 14, 14, 40)        6440
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 14, 40)        160
_________________________________________________________________
leaky_re_lu_17 (LeakyReLU)   (None, 14, 14, 40)        0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 7, 7, 40)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1960)              0
_________________________________________________________________
dense_6 (Dense)              (None, 32)                62752
```

```
_________________________________________________________________
batch_normalization_4 (Batch (None, 32)                128
_________________________________________________________________
leaky_re_lu_18 (LeakyReLU)   (None, 32)                0
_________________________________________________________________
dense_7 (Dense)              (None, 32)                1056
_________________________________________________________________
batch_normalization_5 (Batch (None, 32)                128
_________________________________________________________________
leaky_re_lu_19 (LeakyReLU)   (None, 32)                0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                330
=================================================================
Total params: 78,374
Trainable params: 78,006
Non-trainable params: 368
_________________________________________________________________
```

```
In [20]: history3 = model.fit(x_train, y_train, batch_size = batch, validation_split = 0.3,
                              epochs = epoch, verbose = verbose)

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 11s 266us/sample - loss: 0.4142 - accuracy: 0.8598
Epoch 2/20
42000/42000 [==============================] - 9s 226us/sample - loss: 0.2719 - accuracy: 0.9004
Epoch 3/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.2316 - accuracy: 0.9153
Epoch 4/20
42000/42000 [==============================] - 9s 222us/sample - loss: 0.2046 - accuracy: 0.9258
Epoch 5/20
42000/42000 [==============================] - 9s 222us/sample - loss: 0.1847 - accuracy: 0.9326
Epoch 6/20
42000/42000 [==============================] - 9s 221us/sample - loss: 0.1674 - accuracy: 0.9380
Epoch 7/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.1516 - accuracy: 0.9446
Epoch 8/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.1404 - accuracy: 0.9482
Epoch 9/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.1275 - accuracy: 0.9536
Epoch 10/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.1169 - accuracy: 0.9571
Epoch 11/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.1066 - accuracy: 0.9609
Epoch 12/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.0972 - accuracy: 0.9640
Epoch 13/20
42000/42000 [==============================] - 9s 225us/sample - loss: 0.0929 - accuracy: 0.9659
Epoch 14/20
```

```
42000/42000 [==============================] - 10s 226us/sample - loss: 0.0866 - accuracy: 0.9686
Epoch 15/20
42000/42000 [==============================] - 9s 226us/sample - loss: 0.0802 - accuracy: 0.9702
Epoch 16/20
42000/42000 [==============================] - 10s 229us/sample - loss: 0.0743 - accuracy: 0.9728
Epoch 17/20
42000/42000 [==============================] - 10s 229us/sample - loss: 0.0722 - accuracy: 0.9718
Epoch 18/20
42000/42000 [==============================] - 10s 230us/sample - loss: 0.0650 - accuracy: 0.9764
Epoch 19/20
42000/42000 [==============================] - 10s 240us/sample - loss: 0.0623 - accuracy: 0.9766
Epoch 20/20
42000/42000 [==============================] - 10s 228us/sample - loss: 0.0584 - accuracy: 0.9784
```

```
In [21]: test_loss, test_acc = model.evaluate(x_test, y_test, verbose = 0)

         print('\nTest acccuracy:', test_acc)
```

```
Test acccuracy: 0.9005
```

- Batch norm antes de activación con Adam

```
In [22]: def cnn():
             model = Sequential()

             model.add(Conv2D(20, (3,3), padding = 'same', activation=None, input_shape = input_s
             model.add(BatchNormalization())
             model.add(LeakyReLU())
             model.add(Conv2D(20, (3,3), padding = 'same', activation=None))
             model.add(BatchNormalization())
             model.add(LeakyReLU())
             model.add(MaxPooling2D((2,2)))

             model.add(Conv2D(40, (2,2), padding = 'same', activation=None))
             model.add(BatchNormalization())
             model.add(LeakyReLU())
             model.add(Conv2D(40, (2,2), padding = 'same', activation=None))
             model.add(BatchNormalization())
             model.add(LeakyReLU())
             model.add(MaxPooling2D((2,2)))


             model.add(Flatten())

             model.add(Dense(32, activation = None))
             model.add(BatchNormalization())
             model.add(LeakyReLU())
```

```
    model.add(Dense(32, activation = None))
    model.add(BatchNormalization())
    model.add(LeakyReLU())
    model.add(Dense(10, activation = 'softmax'))

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

```
In [23]: model = cnn()
         model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 28, 28, 20)        200
_________________________________________________________________
batch_normalization_6 (Batch (None, 28, 28, 20)        80
_________________________________________________________________
leaky_re_lu_20 (LeakyReLU)   (None, 28, 28, 20)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 28, 28, 20)        3620
_________________________________________________________________
batch_normalization_7 (Batch (None, 28, 28, 20)        80
_________________________________________________________________
leaky_re_lu_21 (LeakyReLU)   (None, 28, 28, 20)        0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 20)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 14, 14, 40)        3240
_________________________________________________________________
batch_normalization_8 (Batch (None, 14, 14, 40)        160
_________________________________________________________________
leaky_re_lu_22 (LeakyReLU)   (None, 14, 14, 40)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 14, 14, 40)        6440
_________________________________________________________________
batch_normalization_9 (Batch (None, 14, 14, 40)        160
_________________________________________________________________
leaky_re_lu_23 (LeakyReLU)   (None, 14, 14, 40)        0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 7, 7, 40)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1960)              0
```

```
_________________________________________________________________
dense_9 (Dense)              (None, 32)                62752
_________________________________________________________________
batch_normalization_10 (Batc (None, 32)                128
_________________________________________________________________
leaky_re_lu_24 (LeakyReLU)   (None, 32)                0
_________________________________________________________________
dense_10 (Dense)             (None, 32)                1056
_________________________________________________________________
batch_normalization_11 (Batc (None, 32)                128
_________________________________________________________________
leaky_re_lu_25 (LeakyReLU)   (None, 32)                0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                330
=================================================================
Total params: 78,374
Trainable params: 78,006
Non-trainable params: 368
_________________________________________________________________
```

```
In [24]: history4 = model.fit(x_train, y_train, batch_size = batch, validation_split = 0.3,
                              epochs = epoch, verbose = verbose)
```

```
Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 10s 248us/sample - loss: 0.4324 - accuracy: 0.8598
Epoch 2/20
42000/42000 [==============================] - 9s 217us/sample - loss: 0.2686 - accuracy: 0.9045
Epoch 3/20
42000/42000 [==============================] - 10s 228us/sample - loss: 0.2304 - accuracy: 0.9160
Epoch 4/20
42000/42000 [==============================] - 9s 225us/sample - loss: 0.2026 - accuracy: 0.9268
Epoch 5/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.1814 - accuracy: 0.9347
Epoch 6/20
42000/42000 [==============================] - 9s 217us/sample - loss: 0.1659 - accuracy: 0.9401
Epoch 7/20
42000/42000 [==============================] - 9s 218us/sample - loss: 0.1530 - accuracy: 0.9433
Epoch 8/20
42000/42000 [==============================] - 9s 218us/sample - loss: 0.1397 - accuracy: 0.9486
Epoch 9/20
42000/42000 [==============================] - 9s 219us/sample - loss: 0.1276 - accuracy: 0.9534
Epoch 10/20
42000/42000 [==============================] - 9s 219us/sample - loss: 0.1147 - accuracy: 0.9570
Epoch 11/20
42000/42000 [==============================] - 9s 220us/sample - loss: 0.1058 - accuracy: 0.9604
Epoch 12/20
42000/42000 [==============================] - 9s 220us/sample - loss: 0.0975 - accuracy: 0.9642
Epoch 13/20
```

```
42000/42000 [==============================] - 9s 220us/sample - loss: 0.0921 - accuracy: 0.9665
Epoch 14/20
42000/42000 [==============================] - 9s 220us/sample - loss: 0.0832 - accuracy: 0.9695
Epoch 15/20
42000/42000 [==============================] - 9s 219us/sample - loss: 0.0776 - accuracy: 0.9709
Epoch 16/20
42000/42000 [==============================] - 9s 221us/sample - loss: 0.0712 - accuracy: 0.9740
Epoch 17/20
42000/42000 [==============================] - 9s 221us/sample - loss: 0.0682 - accuracy: 0.9761
Epoch 18/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.0647 - accuracy: 0.9764
Epoch 19/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.0586 - accuracy: 0.9784
Epoch 20/20
42000/42000 [==============================] - 9s 222us/sample - loss: 0.0555 - accuracy: 0.9798
```

```
In [25]: test_loss, test_acc = model.evaluate(x_test, y_test, verbose = 0)

         print('\nTest acccuracy:', test_acc)
```

```
Test acccuracy: 0.9076
```

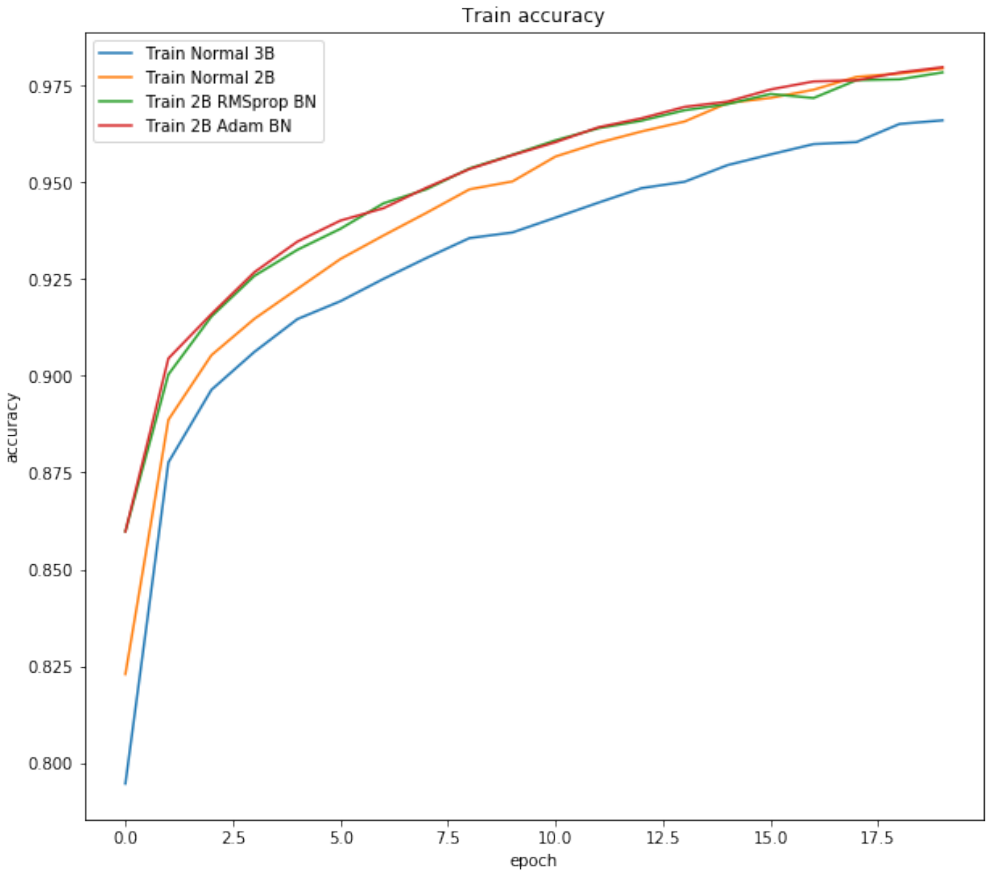
Train metrics

```
In [26]: #plot
         plt.figure(figsize=(10,9))

         plt.plot(history1.history['accuracy'])
         plt.plot(history2.history['accuracy'])
         plt.plot(history3.history['accuracy'])
         plt.plot(history4.history['accuracy'])


         plt.legend(['Train Normal 3B',
                     'Train Normal 2B',
                     'Train 2B RMSprop BN',
                     'Train 2B Adam BN'])

         plt.title('Train accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.show()
```



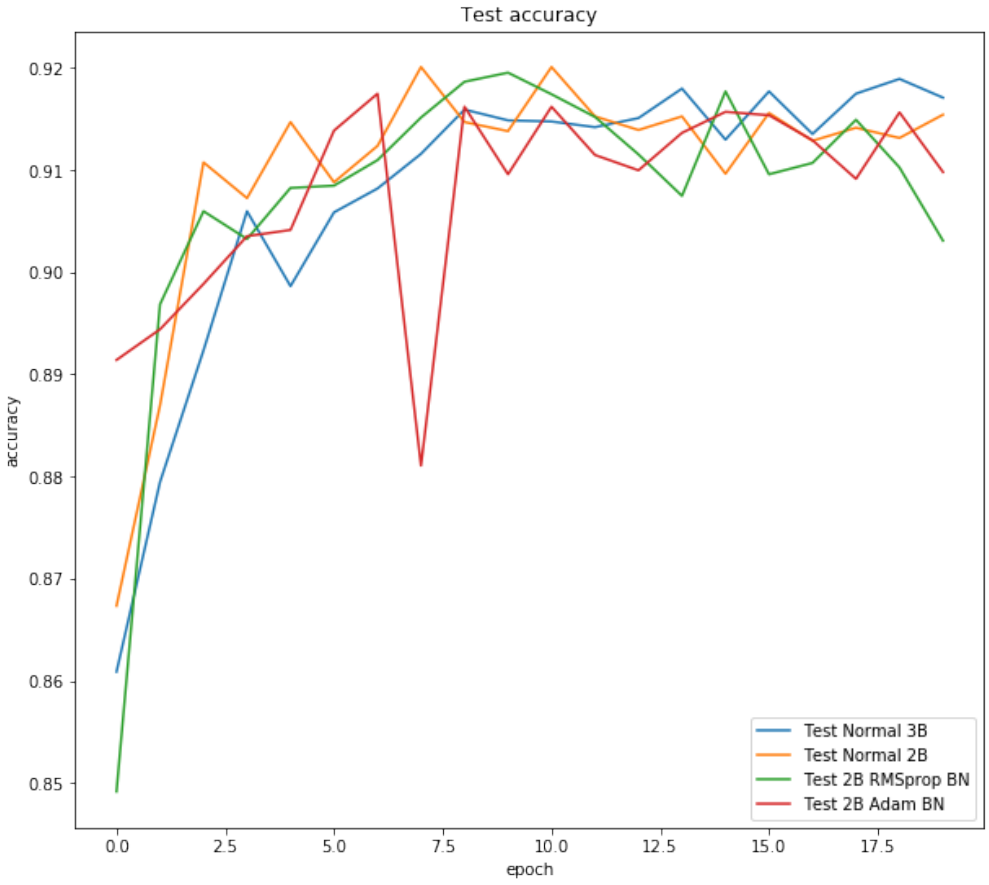
Test metrics

```
In [27]: #plot
         plt.figure(figsize=(10,9))
         plt.plot(history1.history['val_accuracy'])
         plt.plot(history2.history['val_accuracy'])
         plt.plot(history3.history['val_accuracy'])
         plt.plot(history4.history['val_accuracy'])

         plt.legend(['Test Normal 3B',
                     'Test Normal 2B',
                     'Test 2B RMSprop BN',
                     'Test 2B Adam BN'])

         plt.title('Test accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
```

```
plt.show()
```



Batch Normalization después de activación

In [28]:
```
from tensorflow.keras.layers import Activation
```

In [29]:
```
def cnn():
    model = Sequential()

    model.add(Conv2D(20, (3,3), padding = 'same', activation=None, input_shape = input_s
    model.add(LeakyReLU())
    model.add(BatchNormalization())
    model.add(Conv2D(20, (3,3), padding = 'same', activation=None))
    model.add(LeakyReLU())
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2,2)))
```

```
    model.add(Conv2D(40, (2,2), padding = 'same', activation=None))
    model.add(LeakyReLU())
    model.add(BatchNormalization())
    model.add(Conv2D(40, (2,2), padding = 'same', activation=None))
    model.add(LeakyReLU())
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2,2)))


    model.add(Flatten())

    model.add(Dense(32, activation = None))
    model.add(LeakyReLU())
    model.add(BatchNormalization())
    model.add(Dense(32, activation = None))
    model.add(LeakyReLU())
    model.add(BatchNormalization())
    model.add(Dense(10, activation = 'softmax'))

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

```
In [30]: model = cnn()
         model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 28, 28, 20)        200
_________________________________________________________________
leaky_re_lu_26 (LeakyReLU)   (None, 28, 28, 20)        0
_________________________________________________________________
batch_normalization_12 (Batc (None, 28, 28, 20)        80
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 28, 28, 20)        3620
_________________________________________________________________
leaky_re_lu_27 (LeakyReLU)   (None, 28, 28, 20)        0
_________________________________________________________________
batch_normalization_13 (Batc (None, 28, 28, 20)        80
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 14, 14, 20)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 14, 14, 40)        3240
```

```
_________________________________________________________________
leaky_re_lu_28 (LeakyReLU)   (None, 14, 14, 40)        0
_________________________________________________________________
batch_normalization_14 (Batc (None, 14, 14, 40)        160
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 14, 14, 40)        6440
_________________________________________________________________
leaky_re_lu_29 (LeakyReLU)   (None, 14, 14, 40)        0
_________________________________________________________________
batch_normalization_15 (Batc (None, 14, 14, 40)        160
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 7, 7, 40)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1960)              0
_________________________________________________________________
dense_12 (Dense)             (None, 32)                62752
_________________________________________________________________
leaky_re_lu_30 (LeakyReLU)   (None, 32)                0
_________________________________________________________________
batch_normalization_16 (Batc (None, 32)                128
_________________________________________________________________
dense_13 (Dense)             (None, 32)                1056
_________________________________________________________________
leaky_re_lu_31 (LeakyReLU)   (None, 32)                0
_________________________________________________________________
batch_normalization_17 (Batc (None, 32)                128
_________________________________________________________________
dense_14 (Dense)             (None, 10)                330
=================================================================
Total params: 78,374
Trainable params: 78,006
Non-trainable params: 368
_________________________________________________________________
```

```
In [31]: history5 = model.fit(x_train, y_train, batch_size = batch, validation_split = 0.3,
                              epochs = epoch, verbose = verbose)

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 11s 258us/sample - loss: 0.4045 - accuracy: 0.8625
Epoch 2/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.2665 - accuracy: 0.9047
Epoch 3/20
42000/42000 [==============================] - 9s 226us/sample - loss: 0.2294 - accuracy: 0.9187
Epoch 4/20
42000/42000 [==============================] - 9s 218us/sample - loss: 0.2032 - accuracy: 0.9262
Epoch 5/20
42000/42000 [==============================] - 9s 225us/sample - loss: 0.1832 - accuracy: 0.9340
Epoch 6/20
```

```
42000/42000 [==============================] - 10s 229us/sample - loss: 0.1662 - accuracy: 0.9400
Epoch 7/20
42000/42000 [==============================] - 10s 229us/sample - loss: 0.1501 - accuracy: 0.9457
Epoch 8/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.1401 - accuracy: 0.9504
Epoch 9/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.1280 - accuracy: 0.9542
Epoch 10/20
42000/42000 [==============================] - 9s 225us/sample - loss: 0.1168 - accuracy: 0.9583
Epoch 11/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.1073 - accuracy: 0.9609
Epoch 12/20
42000/42000 [==============================] - 9s 222us/sample - loss: 0.1011 - accuracy: 0.9634
Epoch 13/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.0913 - accuracy: 0.9674
Epoch 14/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.0864 - accuracy: 0.9685
Epoch 15/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.0797 - accuracy: 0.9712
Epoch 16/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.0747 - accuracy: 0.9725
Epoch 17/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.0681 - accuracy: 0.9752
Epoch 18/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.0672 - accuracy: 0.9761
Epoch 19/20
42000/42000 [==============================] - 9s 224us/sample - loss: 0.0604 - accuracy: 0.9782
Epoch 20/20
42000/42000 [==============================] - 9s 223us/sample - loss: 0.0578 - accuracy: 0.9790
```

```
In [32]: test_loss, test_acc = model.evaluate(x_test, y_test, verbose = 0)

         print('\nTest acccuracy:', test_acc)
```

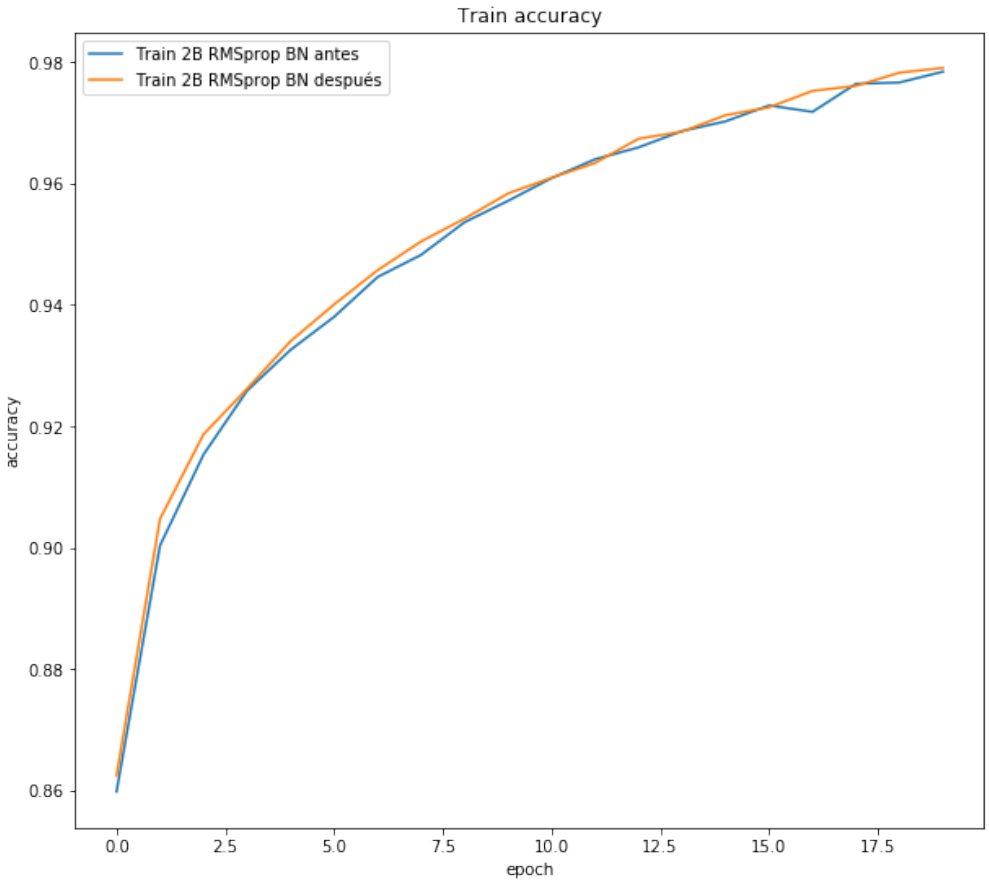
```
Test acccuracy: 0.9083
```

```
In [33]: #plot
         plt.figure(figsize=(10,9))


         plt.plot(history3.history['accuracy'])
         plt.plot(history5.history['accuracy'])


         plt.legend(['Train 2B RMSprop BN antes',
                     'Train 2B RMSprop BN después'])
```

```
plt.title('Train accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.show()
```



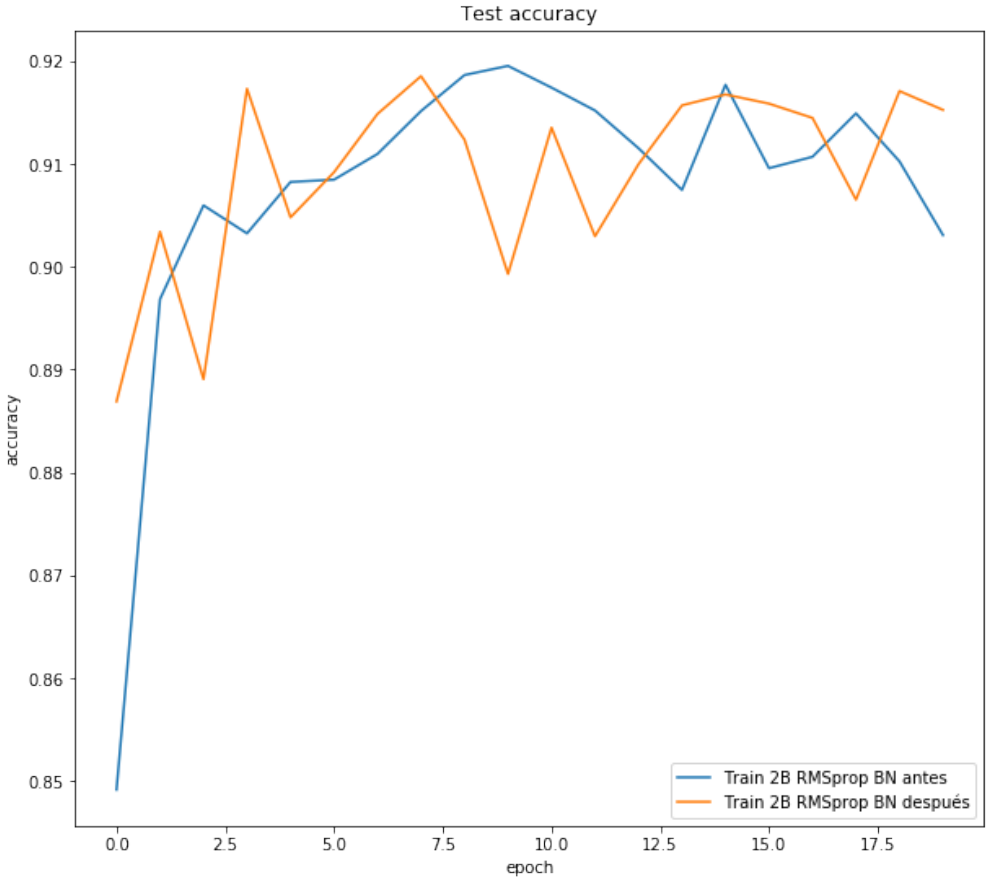
In [34]:
```
#plot
plt.figure(figsize=(10,9))


plt.plot(history3.history['val_accuracy'])
plt.plot(history5.history['val_accuracy'])


plt.legend(['Train 2B RMSprop BN antes',
            'Train 2B RMSprop BN después'])

plt.title('Test accuracy')
plt.ylabel('accuracy')
```

```
plt.xlabel('epoch')
plt.show()
```



- Modificar la arquitectura para entrenar en menos tiempo y obtener mejor test accuracy.
- Agregar otros métodos de regularización
- Experimentar con otro dataset
- Experimentar con el número de filtros, pool_size y kernel_size