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

2019/09/30

1 Redes Neuronales Convolucionales Profundas y Regularización

- Objetivo: Implementar redes convolucionales profundas, conocer el desempeño de los optimizadores y los efectos de regularización y profundidad en el entrenamiento. Se conocerá el resultado de BatchNormalization antes y después de la activación
- Se apilan dos a tres bloques 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)
```

1.1 Obtener dimensiones

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

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

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

1.2 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)
```

```
Epoch 1/20
840/840 [==============================] - 9s 11ms/step - loss: 0.5865 - accuracy: 0.7823 - val_l
Epoch 2/20
840/840 [==============================] - 9s 11ms/step - loss: 0.3574 - accuracy: 0.8690 - val_l
Epoch 3/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2962 - accuracy: 0.8916 - val_l
Epoch 4/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2612 - accuracy: 0.9030 - val_l
Epoch 5/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2370 - accuracy: 0.9135 - val_l
Epoch 6/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2182 - accuracy: 0.9205 - val_l
Epoch 7/20
840/840 [==============================] - 9s 10ms/step - loss: 0.2061 - accuracy: 0.9251 - val_l
Epoch 8/20
```

```
840/840 [==============================] - 9s 10ms/step - loss: 0.1891 - accuracy: 0.9287 - val_l
Epoch 9/20
840/840 [==============================] - 9s 10ms/step - loss: 0.1782 - accuracy: 0.9350 - val_l
Epoch 10/20
840/840 [==============================] - 9s 10ms/step - loss: 0.1655 - accuracy: 0.9377 - val_l
Epoch 11/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1526 - accuracy: 0.9442 - val_l
Epoch 12/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1440 - accuracy: 0.9461 - val_l
Epoch 13/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1370 - accuracy: 0.9495 - val_l
Epoch 14/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1278 - accuracy: 0.9527 - val_l
Epoch 15/20
840/840 [==============================] - 9s 10ms/step - loss: 0.1196 - accuracy: 0.9550 - val_l
Epoch 16/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1110 - accuracy: 0.9585 - val_l
Epoch 17/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1068 - accuracy: 0.9599 - val_l
Epoch 18/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0957 - accuracy: 0.9638 - val_l
Epoch 19/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0907 - accuracy: 0.9664 - val_l
Epoch 20/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0853 - accuracy: 0.9678 - val_l
```

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

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

```
Test acccuracy: 0.910099983215332
```

- 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)
```

```
Epoch 1/20
840/840 [==============================] - 8s 10ms/step - loss: 0.5116 - accuracy: 0.8125 - val_l
Epoch 2/20
840/840 [==============================] - 8s 9ms/step - loss: 0.3214 - accuracy: 0.8846 - val_lo
Epoch 3/20
840/840 [==============================] - 8s 9ms/step - loss: 0.2715 - accuracy: 0.9021 - val_lo
Epoch 4/20
840/840 [==============================] - 8s 9ms/step - loss: 0.2390 - accuracy: 0.9136 - val_lo
Epoch 5/20
840/840 [==============================] - 8s 10ms/step - loss: 0.2141 - accuracy: 0.9220 - val_l
Epoch 6/20
840/840 [==============================] - 8s 9ms/step - loss: 0.1934 - accuracy: 0.9282 - val_lo
Epoch 7/20
840/840 [==============================] - 8s 9ms/step - loss: 0.1739 - accuracy: 0.9357 - val_lo
Epoch 8/20
840/840 [==============================] - 8s 9ms/step - loss: 0.1575 - accuracy: 0.9417 - val_lo
Epoch 9/20
840/840 [==============================] - 8s 10ms/step - loss: 0.1421 - accuracy: 0.9470 - val_l
Epoch 10/20
840/840 [==============================] - 8s 9ms/step - loss: 0.1334 - accuracy: 0.9494 - val_lo
Epoch 11/20
840/840 [==============================] - 8s 9ms/step - loss: 0.1172 - accuracy: 0.9573 - val_lo
Epoch 12/20
840/840 [==============================] - 8s 9ms/step - loss: 0.1073 - accuracy: 0.9602 - val_lo
Epoch 13/20
840/840 [==============================] - 8s 9ms/step - loss: 0.0967 - accuracy: 0.9650 - val_lo
Epoch 14/20
840/840 [==============================] - 8s 9ms/step - loss: 0.0888 - accuracy: 0.9672 - val_lo
Epoch 15/20
840/840 [==============================] - 8s 9ms/step - loss: 0.0799 - accuracy: 0.9706 - val_lo
Epoch 16/20
840/840 [==============================] - 8s 10ms/step - loss: 0.0706 - accuracy: 0.9740 - val_l
Epoch 17/20
840/840 [==============================] - 8s 10ms/step - loss: 0.0656 - accuracy: 0.9754 - val_l
```

```
Epoch 18/20
840/840 [==============================] - 8s 9ms/step - loss: 0.0611 - accuracy: 0.9774 - val_lo
Epoch 19/20
840/840 [==============================] - 8s 9ms/step - loss: 0.0550 - accuracy: 0.9794 - val_lo
Epoch 20/20
840/840 [==============================] - 8s 9ms/step - loss: 0.0543 - accuracy: 0.9805 - val_lo
```

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

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

```
Test acccuracy: 0.904699981212616
```

1.3 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)
```

```
Epoch 1/20
840/840 [==============================] - 9s 11ms/step - loss: 0.4068 - accuracy: 0.8644 - val_l
Epoch 2/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2665 - accuracy: 0.9050 - val_l
Epoch 3/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2292 - accuracy: 0.9170 - val_l
Epoch 4/20
840/840 [==============================] - 10s 11ms/step - loss: 0.2063 - accuracy: 0.9263 - val_
Epoch 5/20
840/840 [==============================] - 10s 11ms/step - loss: 0.1860 - accuracy: 0.9309 - val_
Epoch 6/20
840/840 [==============================] - 10s 11ms/step - loss: 0.1689 - accuracy: 0.9375 - val_
Epoch 7/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1532 - accuracy: 0.9445 - val_l
Epoch 8/20
840/840 [==============================] - 10s 11ms/step - loss: 0.1428 - accuracy: 0.9470 - val_
Epoch 9/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1301 - accuracy: 0.9534 - val_l
Epoch 10/20
840/840 [==============================] - 10s 11ms/step - loss: 0.1206 - accuracy: 0.9562 - val_
Epoch 11/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1118 - accuracy: 0.9594 - val_l
Epoch 12/20
840/840 [==============================] - 10s 11ms/step - loss: 0.1006 - accuracy: 0.9630 - val_
Epoch 13/20
840/840 [==============================] - 10s 11ms/step - loss: 0.0933 - accuracy: 0.9667 - val_
Epoch 14/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0899 - accuracy: 0.9669 - val_l
Epoch 15/20
840/840 [==============================] - 10s 11ms/step - loss: 0.0835 - accuracy: 0.9695 - val_
Epoch 16/20
```

```
840/840 [==============================] - 10s 11ms/step - loss: 0.0761 - accuracy: 0.9723 - val_
Epoch 17/20
840/840 [==============================] - 10s 11ms/step - loss: 0.0729 - accuracy: 0.9737 - val_
Epoch 18/20
840/840 [==============================] - 10s 12ms/step - loss: 0.0690 - accuracy: 0.9755 - val_
Epoch 19/20
840/840 [==============================] - 10s 11ms/step - loss: 0.0649 - accuracy: 0.9771 - val_
Epoch 20/20
840/840 [==============================] - 10s 11ms/step - loss: 0.0632 - accuracy: 0.9775 - val_
```

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

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

```
Test acccuracy: 0.9035999774932861
```

- 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)
```

```
Epoch 1/20
840/840 [==============================] - 9s 11ms/step - loss: 0.4374 - accuracy: 0.8589 - val_l
Epoch 2/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2656 - accuracy: 0.9046 - val_l
Epoch 3/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2246 - accuracy: 0.9193 - val_l
Epoch 4/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1979 - accuracy: 0.9280 - val_l
Epoch 5/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1812 - accuracy: 0.9352 - val_l
Epoch 6/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1651 - accuracy: 0.9390 - val_l
Epoch 7/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1470 - accuracy: 0.9457 - val_l
Epoch 8/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1357 - accuracy: 0.9508 - val_l
Epoch 9/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1233 - accuracy: 0.9548 - val_l
Epoch 10/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1146 - accuracy: 0.9577 - val_l
Epoch 11/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1033 - accuracy: 0.9618 - val_l
Epoch 12/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0975 - accuracy: 0.9633 - val_l
Epoch 13/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0854 - accuracy: 0.9676 - val_l
Epoch 14/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0794 - accuracy: 0.9703 - val_l
Epoch 15/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0732 - accuracy: 0.9727 - val_l
```

```
Epoch 16/20
840/840 [==============================] - 10s 11ms/step - loss: 0.0693 - accuracy: 0.9748 - val_
Epoch 17/20
840/840 [==============================] - 10s 12ms/step - loss: 0.0648 - accuracy: 0.9761 - val_
Epoch 18/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0608 - accuracy: 0.9775 - val_l
Epoch 19/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0561 - accuracy: 0.9807 - val_l
Epoch 20/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0527 - accuracy: 0.9803 - val_l
```

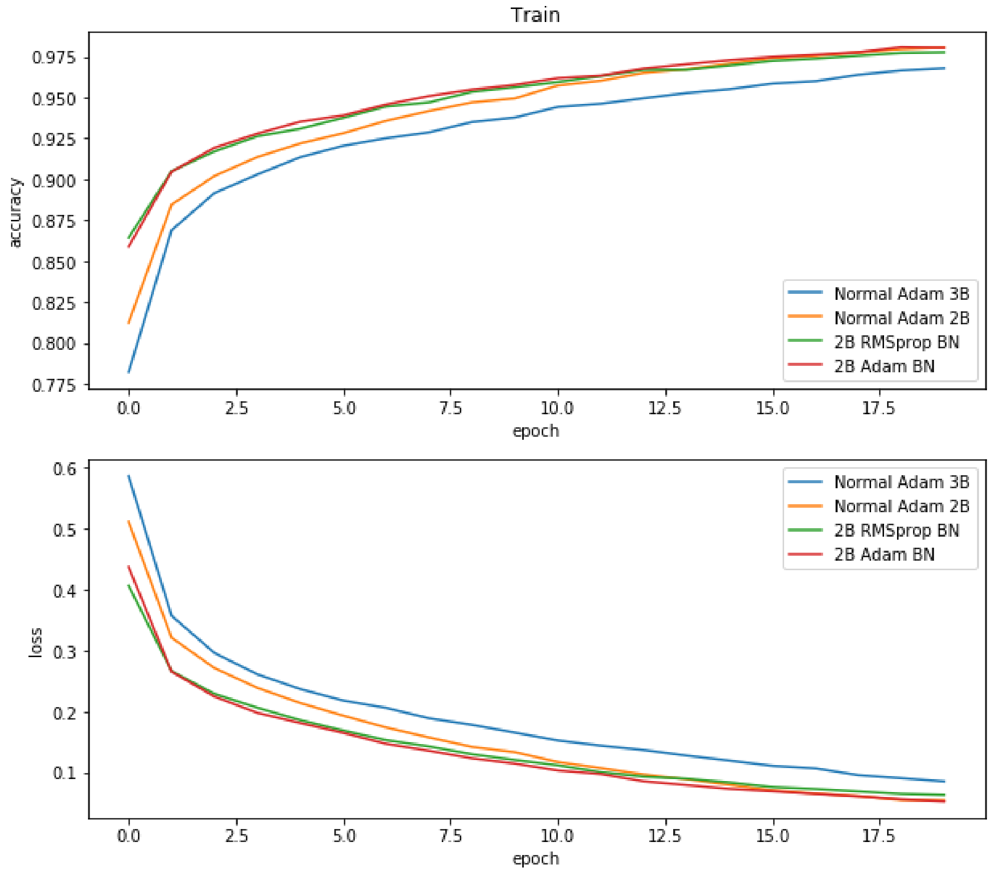
In [26]:
```python
test_loss, test_acc = model.evaluate(x_test, y_test, verbose = 0)

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

```
Test acccuracy: 0.9103999733924866
```

1.4 Plots de entrenamiento

In [49]:
```python
#plot
plt.figure(figsize=(10,9))
plt.subplot(211)
plt.plot(history1.history['accuracy'])
plt.plot(history2.history['accuracy'])
plt.plot(history3.history['accuracy'])
plt.plot(history4.history['accuracy'])


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

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

plt.subplot(212)
plt.plot(history1.history['loss'])
plt.plot(history2.history['loss'])
plt.plot(history3.history['loss'])
plt.plot(history4.history['loss'])


plt.legend(['Normal Adam 3B',
            'Normal Adam 2B',
            '2B RMSprop BN',
```

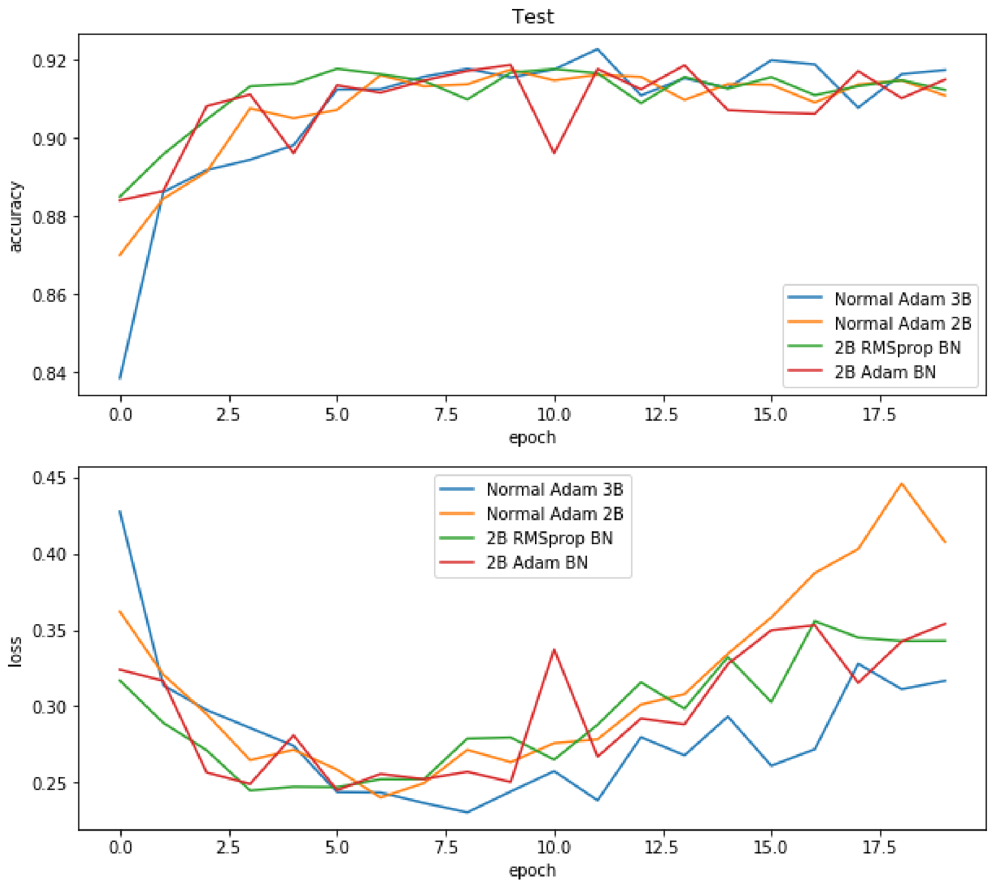
```
                '2B Adam BN'])


plt.ylabel('loss')
plt.xlabel('epoch')

plt.show()
```



1.5 Plots de validación

```
In [50]: #plot
         plt.figure(figsize=(10,9))
         plt.subplot(211)
         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(['Normal Adam 3B',
            'Normal Adam 2B',
            '2B RMSprop BN',
            '2B Adam BN'])

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

plt.subplot(212)
plt.plot(history1.history['val_loss'])
plt.plot(history2.history['val_loss'])
plt.plot(history3.history['val_loss'])
plt.plot(history4.history['val_loss'])

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


plt.ylabel('loss')
plt.xlabel('epoch')

plt.show()
```



- Usar EarlyStopping

1.6 Batch Normalization después de activación

```
In [35]: 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())
```

```python
    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 [36]:
```python
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 [37]: history5 = model.fit(x_train, y_train, batch_size = batch, validation_split = 0.3,
                              epochs = epoch, verbose = verbose)
```

```
Epoch 1/20
840/840 [==============================] - 10s 12ms/step - loss: 0.4002 - accuracy: 0.8617 - val_
Epoch 2/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2685 - accuracy: 0.9045 - val_l
Epoch 3/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2270 - accuracy: 0.9177 - val_l
Epoch 4/20
840/840 [==============================] - 9s 11ms/step - loss: 0.2024 - accuracy: 0.9275 - val_l
Epoch 5/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1834 - accuracy: 0.9336 - val_l
Epoch 6/20
840/840 [==============================] - 10s 12ms/step - loss: 0.1640 - accuracy: 0.9400 - val_
Epoch 7/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1511 - accuracy: 0.9464 - val_l
Epoch 8/20
```

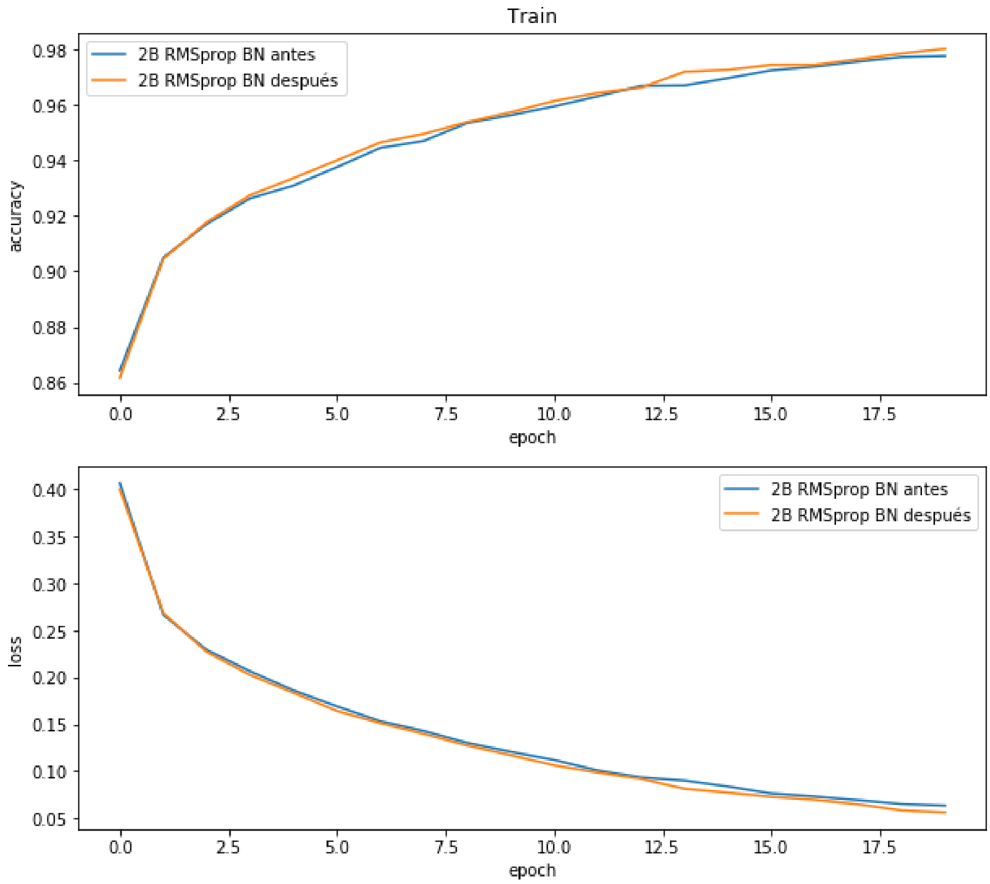
```
840/840 [==============================] - 9s 11ms/step - loss: 0.1397 - accuracy: 0.9495 - val_l
Epoch 9/20
840/840 [==============================] - 9s 11ms/step - loss: 0.1272 - accuracy: 0.9538 - val_l
Epoch 10/20
840/840 [==============================] - 10s 12ms/step - loss: 0.1169 - accuracy: 0.9573 - val_
Epoch 11/20
840/840 [==============================] - 10s 12ms/step - loss: 0.1062 - accuracy: 0.9613 - val_
Epoch 12/20
840/840 [==============================] - 10s 12ms/step - loss: 0.0983 - accuracy: 0.9642 - val_
Epoch 13/20
840/840 [==============================] - 10s 12ms/step - loss: 0.0918 - accuracy: 0.9661 - val_
Epoch 14/20
840/840 [==============================] - 10s 12ms/step - loss: 0.0811 - accuracy: 0.9718 - val_
Epoch 15/20
840/840 [==============================] - 10s 11ms/step - loss: 0.0770 - accuracy: 0.9725 - val_
Epoch 16/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0725 - accuracy: 0.9743 - val_l
Epoch 17/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0694 - accuracy: 0.9743 - val_l
Epoch 18/20
840/840 [==============================] - 9s 11ms/step - loss: 0.0646 - accuracy: 0.9763 - val_l
Epoch 19/20
840/840 [==============================] - 10s 12ms/step - loss: 0.0582 - accuracy: 0.9784 - val_
Epoch 20/20
840/840 [==============================] - 10s 11ms/step - loss: 0.0560 - accuracy: 0.9801 - val_
```

```
In [38]: test_loss, test_acc = model.evaluate(x_test, y_test, verbose = 0)

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

```
Test acccuracy: 0.9043999910354614
```

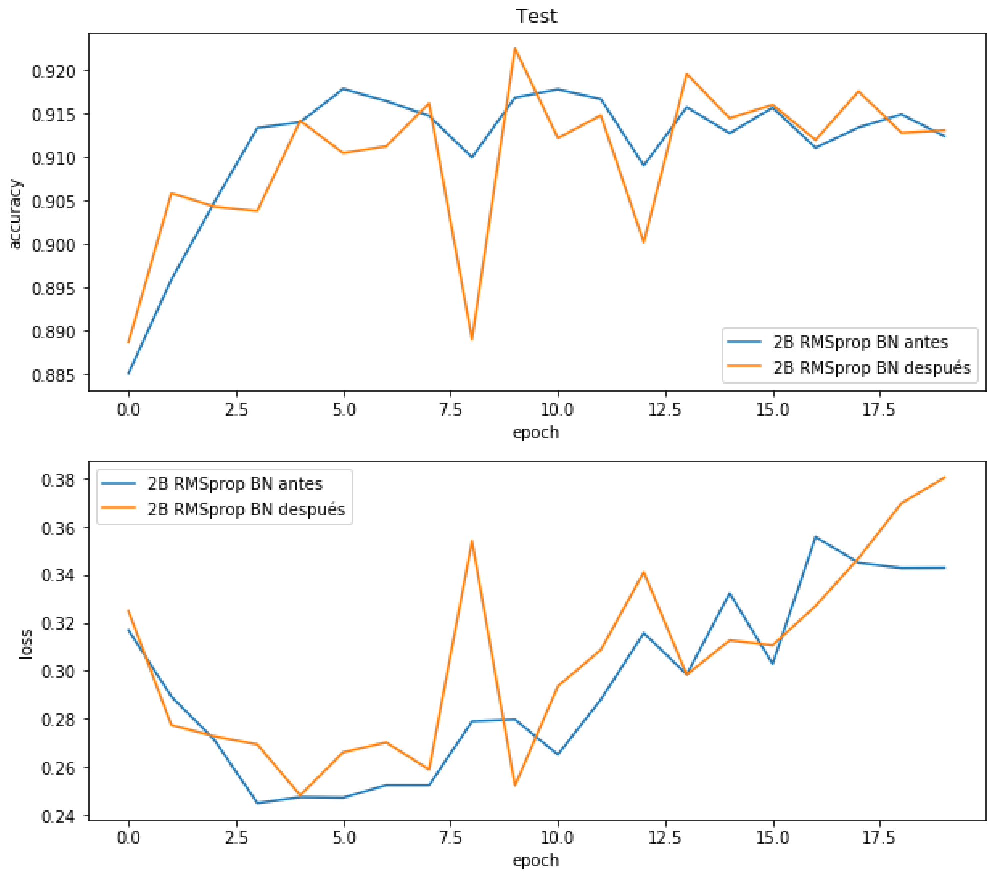
```
In [51]: #plot
         plt.figure(figsize=(10,9))
         plt.subplot(211)

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


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

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

```
plt.subplot(212)
plt.plot(history3.history['loss'])
plt.plot(history5.history['loss'])


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

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




```
In [58]: #plot
         plt.figure(figsize=(10,9))
         plt.subplot(211)

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

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


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

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


plt.subplot(212)
plt.plot(history3.history['val_loss'])
plt.plot(history5.history['val_loss'])


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

plt.ylabel('loss')
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, pooling, strides y kernel_size