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

2019/09/30

Redes Neuronales Convolucionales (CNN)

- Obetivo: Conocer el tipo de capas de las CNN.
- Las redes convolucionales no solo se aplican a imágenes, también se pueden aplicar a caracteres o datos en el tiempo.

Convolución

- La convolución es una operación matemática generalmente denotada como $*$, en la que una función se aplica otra función, dando como resultado la combinación de las dos funciones.

Clasificar ropa

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

        import tensorflow as tf
        from tensorflow import keras
        from tensorflow.keras import backend as K
        K.clear_session()

        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]: # con 1: no se cuenta la primera dimensión
        x, y, channel = x_train.shape[1:]

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

Crear modelo

```
In [6]: from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D
```

```
In [7]: epoch = 5
        verbose = 1
        batch = 50
```

Capa de convolución

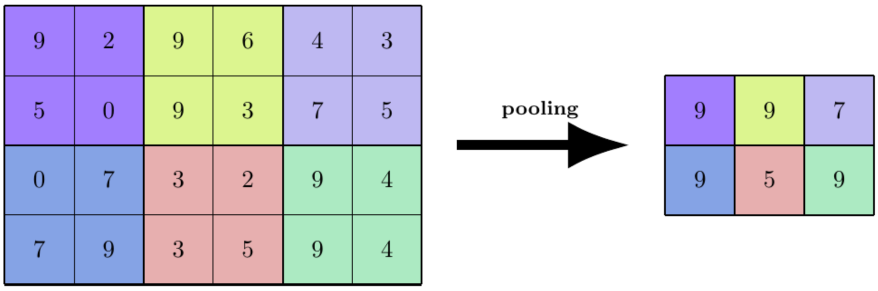
- En general se utiliza la convolución 2D para el procesamiento de imagenes
- En tamaño del filtro es igual al ancho y largo de los campos receptivos
- Pooling reduce el número de parámetros

```
In [8]: num_filters = 20
        filter_size = 3
        pool_size = 3
```

- Diferente notación para crear modelo

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 4 & 2 & \boxed{5} \\ 1 & 2 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

Convolución



Pooling

```
In [9]: model = Sequential([
            Conv2D(num_filters, filter_size, input_shape=input_shape),
            MaxPooling2D(pool_size=pool_size),
            Flatten(),
            Dense(10, activation='softmax'),
        ])
```

Compilar

```
In [10]: model.compile('adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
In [11]: model.summary()
```

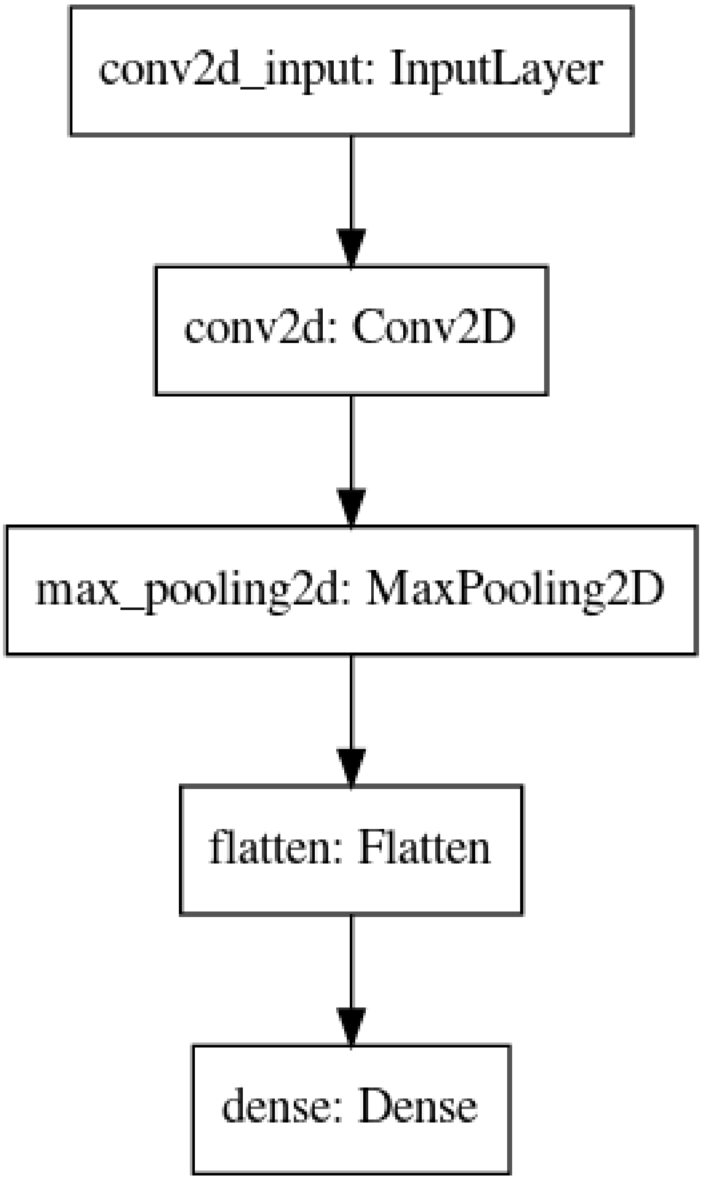
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 20)        200
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 8, 8, 20)          0
_________________________________________________________________
flatten (Flatten)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 10)                12810
=================================================================
Total params: 13,010
Trainable params: 13,010
```

```
Non-trainable params: 0
_________________________________________________________________
```

```
In [12]: from tensorflow.keras.utils import plot_model
```

```
In [13]: plot_model(model)
```

Out[13]:



Entrenamiento

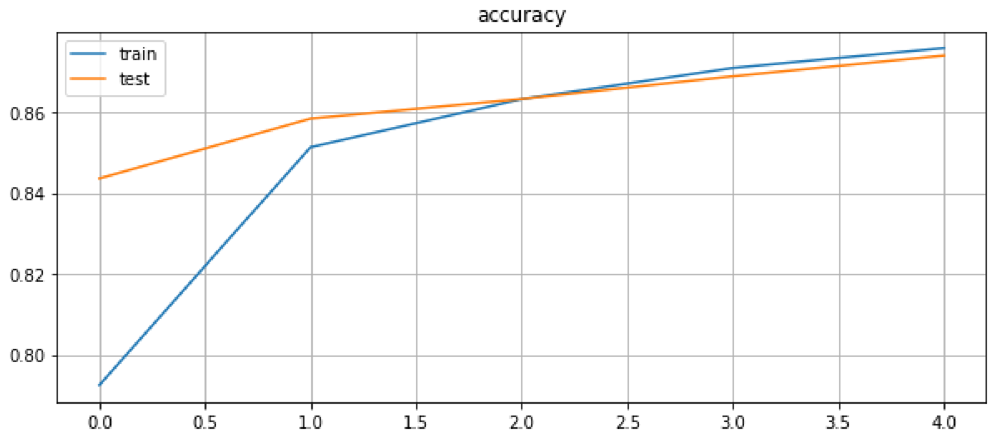
```
In [14]: history = model.fit(x_train, y_train,
           batch_size = batch,
           validation_split=0.3,
           epochs=epoch, verbose = verbose)
```

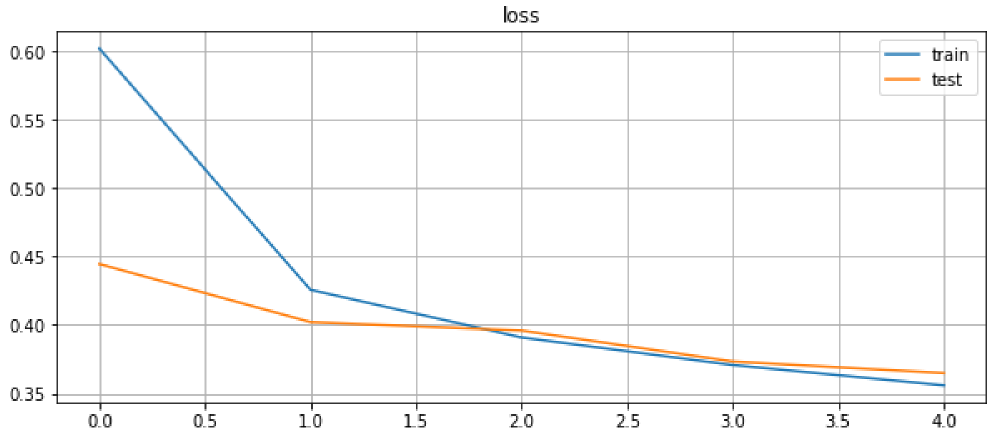
```
Train on 42000 samples, validate on 18000 samples
Epoch 1/5
42000/42000 [==============================] - 5s 116us/sample - loss: 0.6021 - accuracy: 0.7926
Epoch 2/5
42000/42000 [==============================] - 3s 60us/sample - loss: 0.4257 - accuracy: 0.8513 -
Epoch 3/5
42000/42000 [==============================] - 3s 60us/sample - loss: 0.3909 - accuracy: 0.8632 -
Epoch 4/5
42000/42000 [==============================] - 3s 60us/sample - loss: 0.3708 - accuracy: 0.8709 -
Epoch 5/5
42000/42000 [==============================] - 2s 59us/sample - loss: 0.3559 - accuracy: 0.8758 -
```

```
In [15]: test_loss, test_acc = model.evaluate(x_test, y_test, verbose = 0)

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

```
Test acccuracy: 0.8669
```

```
In [16]: #plot
         plt.figure(figsize=(10,9))

         plt.subplot(211)
         plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title('accuracy')
         plt.legend(['train', 'test'])
         plt.grid()
         plt.subplot(212)
         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('loss')
         plt.legend(['train', 'test'])
         plt.grid()




         plt.show()
```





Diferente número de filtros

- Observar el número de parámetros y el efecto de los filtros y pooling en ‘test accuracy’

In [17]:
```
num_filters = 30
filter_size = 3
pool_size = 3

model = Sequential([
  Conv2D(num_filters, filter_size, input_shape=input_shape),
  MaxPooling2D(pool_size=pool_size),
  Flatten(),
  Dense(10, activation='softmax'),
])

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

model.summary()
```

```
history = model.fit(x_train, y_train,
  batch_size = batch,
  validation_split=0.3,
  epochs=epoch, verbose = verbose)
```

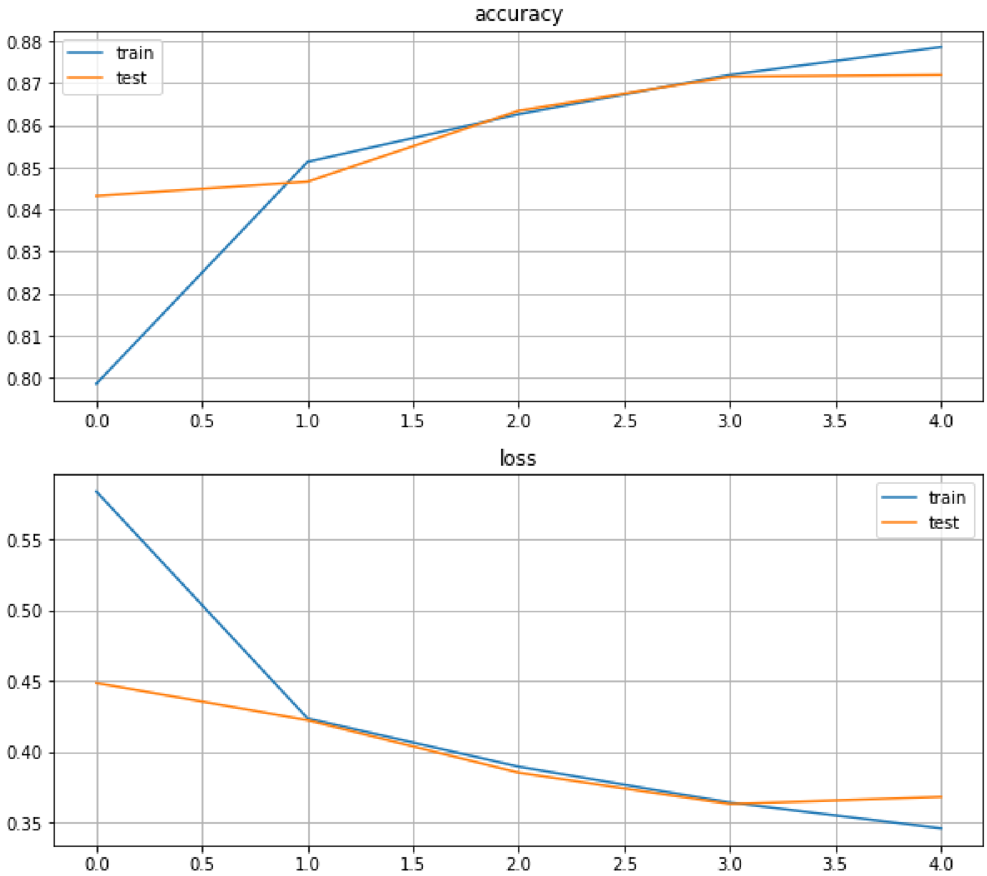
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 30)        300       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 30)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 1920)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 10)                19210     
=================================================================
Total params: 19,510
Trainable params: 19,510
Non-trainable params: 0
_________________________________________________________________
Train on 42000 samples, validate on 18000 samples
Epoch 1/5
42000/42000 [==============================] - 3s 70us/sample - loss: 0.5839 - accuracy: 0.7986 -
Epoch 2/5
42000/42000 [==============================] - 3s 63us/sample - loss: 0.4235 - accuracy: 0.8513 -
Epoch 3/5
42000/42000 [==============================] - 3s 64us/sample - loss: 0.3893 - accuracy: 0.8626 -
Epoch 4/5
42000/42000 [==============================] - 3s 65us/sample - loss: 0.3639 - accuracy: 0.8720 -
Epoch 5/5
42000/42000 [==============================] - 3s 64us/sample - loss: 0.3457 - accuracy: 0.8786 -
```

In [18]:
```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose = 0)

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

```
Test acccuracy: 0.8645
```

In [19]:
```
#plot
plt.figure(figsize=(10,9))

plt.subplot(211)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('accuracy')
```

```
plt.legend(['train', 'test'])
plt.grid()
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('loss')
plt.legend(['train', 'test'])
plt.grid()



plt.show()
```



DIferente tamaño de pooling

```
In [20]: num_filters = 30
         filter_size = 3
```

```
pool_size = 2

model = Sequential([
  Conv2D(num_filters, filter_size, input_shape=input_shape),
  MaxPooling2D(pool_size=pool_size),
  Flatten(),
  Dense(10, activation='softmax'),
])

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

model.summary()

history = model.fit(
  x_train, y_train,
  batch_size = batch,
  validation_split=0.3,
  epochs=epoch, verbose = verbose)
```

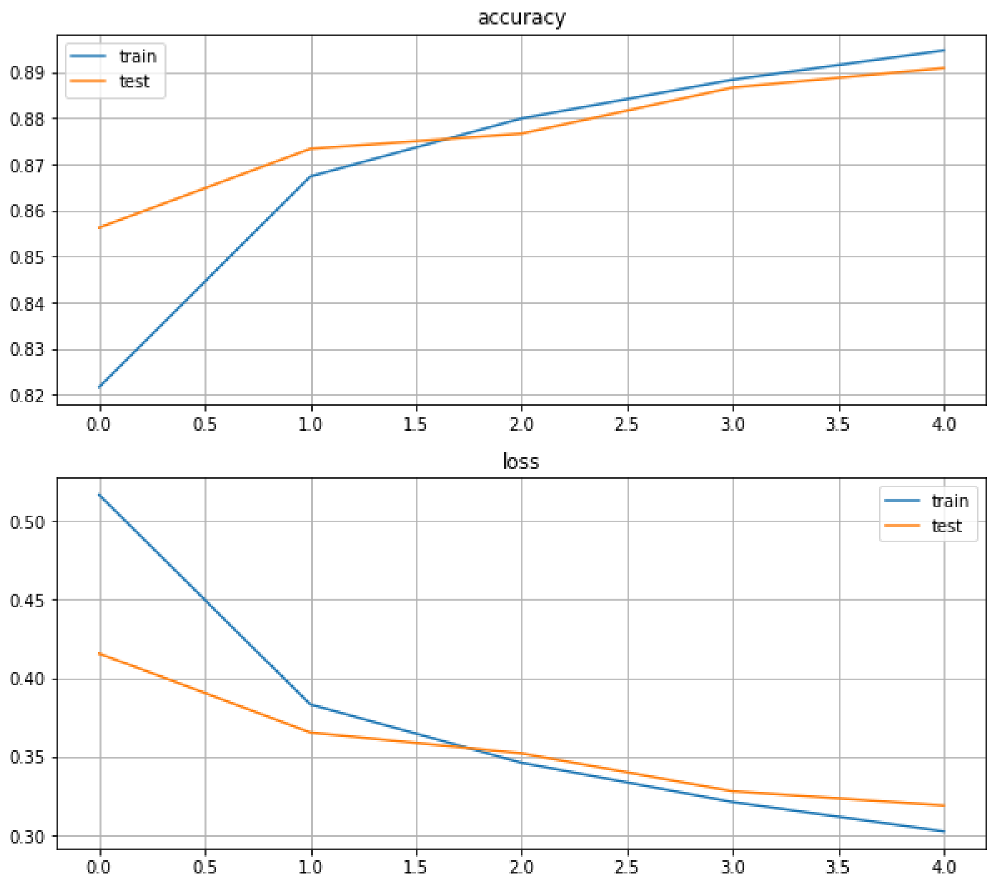
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 30)        300
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 30)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 5070)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                50710
=================================================================
Total params: 51,010
Trainable params: 51,010
Non-trainable params: 0
_________________________________________________________________
Train on 42000 samples, validate on 18000 samples
Epoch 1/5
42000/42000 [==============================] - 3s 75us/sample - loss: 0.5164 - accuracy: 0.8217 -
Epoch 2/5
42000/42000 [==============================] - 3s 71us/sample - loss: 0.3831 - accuracy: 0.8673 -
Epoch 3/5
42000/42000 [==============================] - 3s 70us/sample - loss: 0.3460 - accuracy: 0.8799 -
Epoch 4/5
42000/42000 [==============================] - 3s 71us/sample - loss: 0.3211 - accuracy: 0.8883 -
Epoch 5/5
42000/42000 [==============================] - 3s 73us/sample - loss: 0.3025 - accuracy: 0.8947 -
```

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

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

```
Test acccuracy: 0.8782
```

In [22]:
```
#plot
plt.figure(figsize=(10,9))

plt.subplot(211)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('accuracy')
plt.legend(['train', 'test'])
plt.grid()
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('loss')
plt.legend(['train', 'test'])
plt.grid()




plt.show()
```



Probar predicciones del modelo

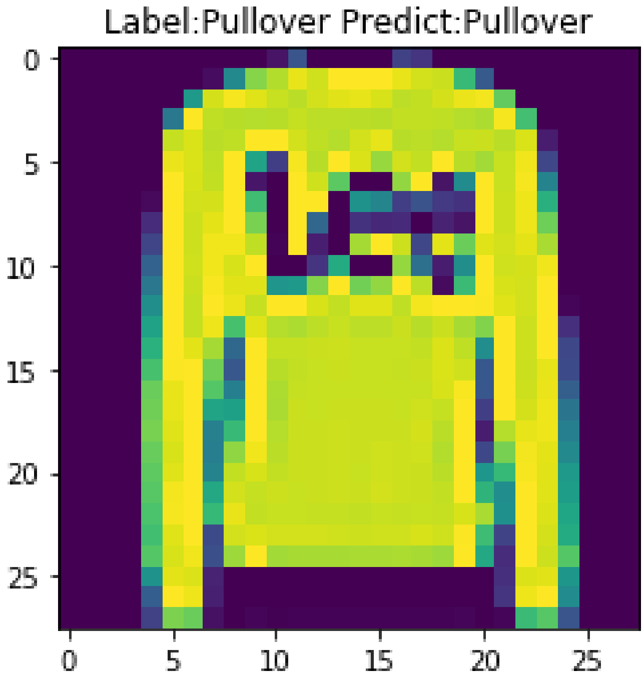
- ‘argmax’ retorna el elemento de mayor valor

```
In [23]: # Primeras 5 imagenes de test
         predictions = model.predict(x_test[:5])

         print(np.argmax(predictions, axis=1))
         p = np.argmax(predictions, axis=1)

         print(y_test[:5])
```

```
[9 2 1 1 6]
[9 2 1 1 6]
```

```
In [24]: plt.imshow(np.squeeze(x_test[1]))
         plt.title('Label:' + class_names[int(y_test[1])]
```

```
             + ' Predict:'+  class_names[int(p[1])])
plt.show()
```



- Mejorar la arquitectura
- Probar con otro dataset