Deep_L_Act_1

October 5, 2023

Estefanía Pérez Yeo - A01639270

1 Cifar10

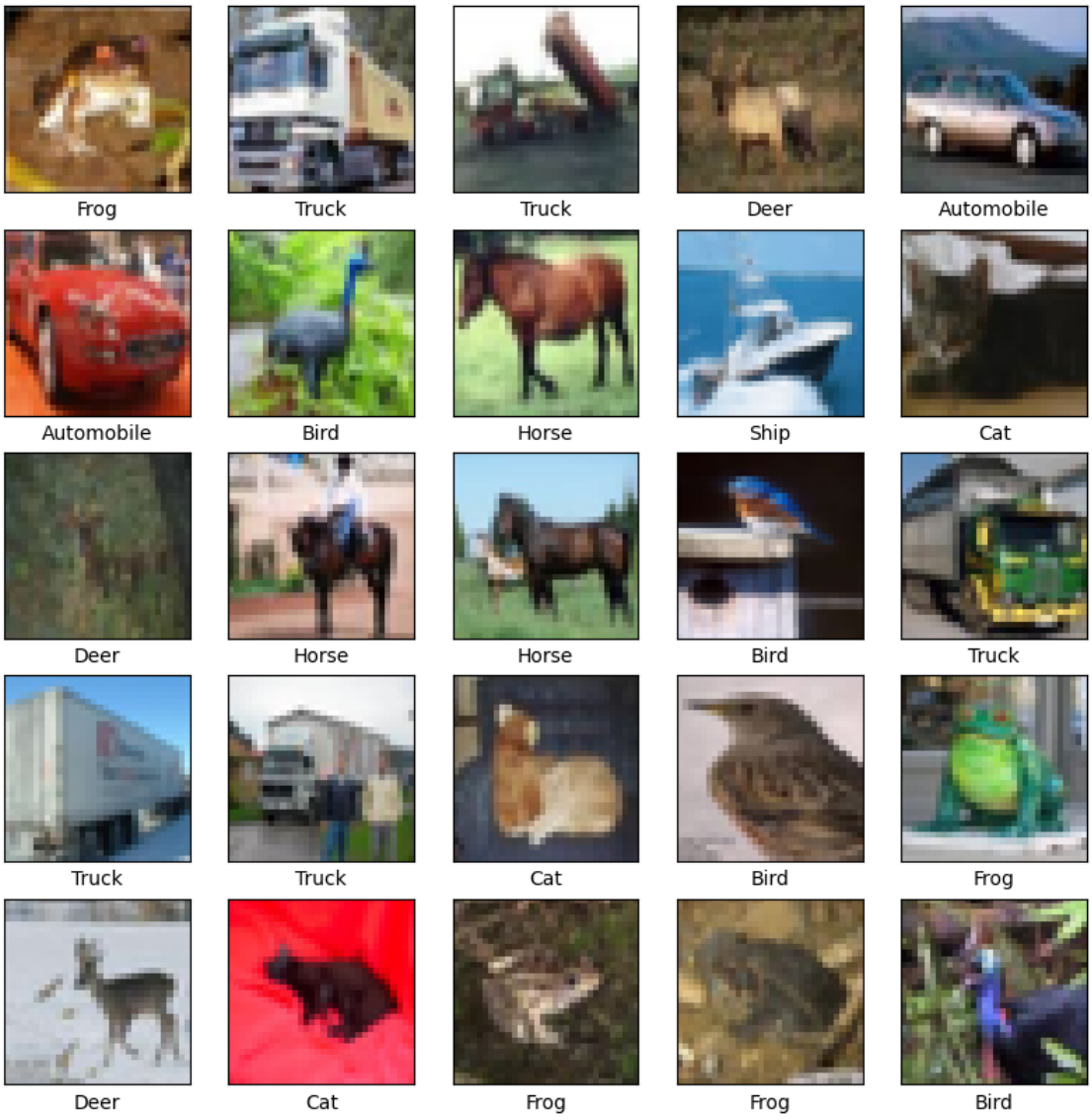
[103]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

[104]:
```
#importar el dataset CIFAR-10
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.
 ↪load_data()

#normlizar las imagenes oara simplificar el procesamiento de la red neuronal␣
 ↪(cambiar los pixeles a valores entre 0 y 1)
train_images, test_images = train_images/255.0, test_images/255.0
```

1.1 Validacion de los datos

[105]:
```
# nombrar las etiquetas de las predicciones para simplificar la interpretacion␣
 ↪de los resultados
class_names = ['Airplane','Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog',␣
 ↪'Horse', 'Ship', 'Truck' ]
```

[106]:
```
plt.figure(figsize=(10,10))
for i in range(25):
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(train_images[i])
  plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```



1.2 Capas de convulación

[135]:
```
model = models.Sequential()
#parametros de: Conv2D(filtros, kernel size, funcion de activacion, tamaño de␣
 ↪input y cuantos canales)

model.add(layers.Conv2D(64,(3,3),activation = 'relu', input_shape=(32,32,3)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128,(3,3),activation = 'relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(256,(3,3),activation = 'relu'))
```

1.3 Arquitectura

[136]:
```
model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_40 (Conv2D)          (None, 30, 30, 64)        1792

 max_pooling2d_27 (MaxPooli  (None, 15, 15, 64)        0
 ng2D)

 conv2d_41 (Conv2D)          (None, 13, 13, 128)       73856

 max_pooling2d_28 (MaxPooli  (None, 6, 6, 128)         0
 ng2D)

 conv2d_42 (Conv2D)          (None, 4, 4, 256)         295168

=================================================================
Total params: 370816 (1.41 MB)
Trainable params: 370816 (1.41 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

1.4 Capas densas

[137]:
```
# tenemos que hacer la salida de 4x4x256 a una sola dimension
```

[138]:
```
model.add(layers.Flatten())
# 64 neuronas, el numero de neuronas varia la presicion de la red neuronal
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(10, activation = 'sigmoid')) # 10 canales de salida porq␣
 ↪hay 10 clases, acotar los numeros entre 0 y 1
```

[139]:
```
# ver la arquitectura de la red oara comprobar que este bien construida
model.summary()

## 64 por 4 por 4 da 1024
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_40 (Conv2D)          (None, 30, 30, 64)        1792

 max_pooling2d_27 (MaxPooli  (None, 15, 15, 64)        0
 ng2D)
```

```
 conv2d_41 (Conv2D)          (None, 13, 13, 128)       73856

 max_pooling2d_28 (MaxPooli  (None, 6, 6, 128)         0
 ng2D)

 conv2d_42 (Conv2D)          (None, 4, 4, 256)         295168

 flatten_14 (Flatten)        (None, 4096)              0

 dense_32 (Dense)            (None, 64)                262208

 dense_33 (Dense)            (None, 10)                650

=================================================================
Total params: 633674 (2.42 MB)
Trainable params: 633674 (2.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

1.5 Entrenamiento

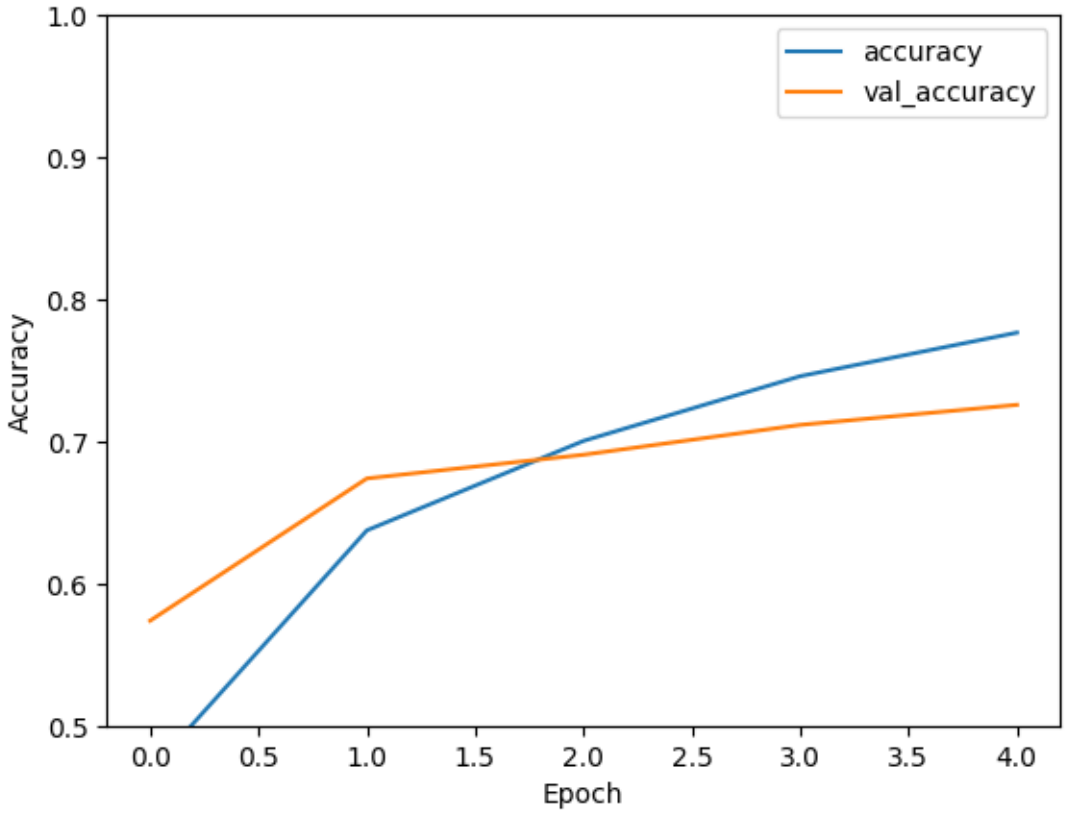
[140]:
```
from matplotlib.rcsetup import validate_font_properties
model.compile(optimizer = 'adam',
              loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits =␣
 ↪True),
              metrics = ['accuracy'])

# numero de epocas podemos jugar para mejorar la presicion
history = model.fit(train_images, train_labels, epochs = 5, validation_data =␣
 ↪(test_images, test_labels))
```

```
Epoch 1/5
1563/1563 [==============================] - 11s 5ms/step - loss: 1.4604 -
accuracy: 0.4677 - val_loss: 1.2244 - val_accuracy: 0.5738
Epoch 2/5
1563/1563 [==============================] - 9s 6ms/step - loss: 1.0322 -
accuracy: 0.6374 - val_loss: 0.9332 - val_accuracy: 0.6739
Epoch 3/5
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8544 -
accuracy: 0.7003 - val_loss: 0.8909 - val_accuracy: 0.6906
Epoch 4/5
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7314 -
accuracy: 0.7459 - val_loss: 0.8395 - val_accuracy: 0.7116
Epoch 5/5
1563/1563 [==============================] - 9s 5ms/step - loss: 0.6367 -
accuracy: 0.7765 - val_loss: 0.8286 - val_accuracy: 0.7257
```

1.6 Evaluacion

[141]:
```
plt.plot(history.history['accuracy'],label = 'accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.ylim([0.5,1])
```

[141]: (0.5, 1.0)



[142]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.8286 - accuracy: 0.7257 - 769ms/epoch - 2ms/step
```

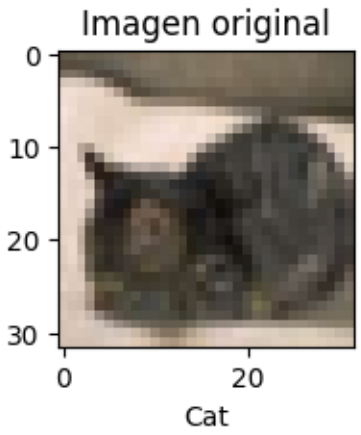
[143]:
```
print("Accuracy: ", test_acc)
```

```
Accuracy:  0.7257000207901001
```

1.7 Predecir

dentro de model despues del fit ahora estan los pesos y las estructuras

[144]:
```
n= 115 #numero de imagen (total de 1000 imagenes)

plt.figure(figsize= (2,2))
plt.imshow(test_images[n])
plt.xlabel(class_names[test_labels[n][0]])
plt.title('Imagen original')
plt.show()
```



[145]:
```
predictions = model.predict(test_images)
print(predictions[n]) #probabilidades de cada una de las neruonas en la ultima␣
↪capa
```

```
313/313 [==============================] - 1s 2ms/step
[0.43494222 0.01087931 0.7099051  0.9741867  0.65115905 0.7156717
 0.1395326  0.5821535  0.00905532 0.02709975]
```

[146]:
```
import numpy as np
print("La imagen pertence al grupo {} con una probabilidad de {:.2f} %"
.format(class_names[np.argmax(predictions[n])], 100 + np.max(predictions[n])))
```

```
La imagen pertence al grupo Cat con una probabilidad de 100.97 %
```

2 Fashion_mnist

[88]:
```
#importar el dataset CIFAR-10
(train_images, train_labels), (test_images, test_labels) = datasets.
↪fashion_mnist.load_data()

#normlizar las imagenes oara simplificar el procesamiento de la red neuronal␣
↪(cambiar los pixeles a valores entre 0 y 1)
train_images, test_images = train_images/255.0, test_images/255.0
```

2.1 Validación de los datos

[89]:
```
# nombrar las etiquetas de las predicciones para simplificar la interpretacion␣
↪de los resultados
class_names = ['T-shirt/top','Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal',␣
↪'Shirt', 'Sneaker', 'Bag', 'Ankle boot' ]
```

[90]:
```
plt.figure(figsize=(10,10))
for i in range(25):
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(train_images[i])
  plt.xlabel(class_names[train_labels[i]])
plt.show()
```

2.2 Capas de convulación

[91]:
```
model = models.Sequential()
#parametros de: Conv2D(filtros, kernel size, funcion de activacion, tamaño de␣
 ↪input y cuantos canales)
# 1 canal porque las imagenes están en grayscale
model.add(layers.Conv2D(64,(3,3),activation = 'relu', input_shape=(28,28,1)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128,(3,3),activation = 'relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128,(3,3),activation = 'relu'))
```

2.3 Arquitectura

[92]:
```
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 26, 26, 64)        640

 max_pooling2d_16 (MaxPooli  (None, 13, 13, 64)        0
 ng2D)

 conv2d_25 (Conv2D)          (None, 11, 11, 128)       73856

 max_pooling2d_17 (MaxPooli  (None, 5, 5, 128)         0
 ng2D)

 conv2d_26 (Conv2D)          (None, 3, 3, 128)         147584

=================================================================
Total params: 222080 (867.50 KB)
Trainable params: 222080 (867.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

2.4 Capas densas

[93]:
```
# tenemos que hacer la salida de 3x3x128 a una sola dimension
```

[94]:
```
model.add(layers.Flatten())
# 64 neuronas porq hay 128 filtros, el numero de neuronas varia la presicion de␣
 ↪la red neuronal
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(10, activation = 'sigmoid')) # 10 canales de salida porq␣
 ↪hay 10 clases, acotar los numeros entre 0 y 1
```

[95]:
```
# ver la arquitectura de la red oara comprobar que este bien construida
model.summary()

## 128 por 4 por 4 da 2048
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 26, 26, 64)        640
```

```
 max_pooling2d_16 (MaxPooli  (None, 13, 13, 64)        0
 ng2D)

 conv2d_25 (Conv2D)          (None, 11, 11, 128)       73856

 max_pooling2d_17 (MaxPooli  (None, 5, 5, 128)         0
 ng2D)

 conv2d_26 (Conv2D)          (None, 3, 3, 128)         147584

 flatten_9 (Flatten)         (None, 1152)              0

 dense_18 (Dense)            (None, 64)                73792

 dense_19 (Dense)            (None, 10)                650

=================================================================
Total params: 296522 (1.13 MB)
Trainable params: 296522 (1.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

2.5 Entrenamiento

[96]:
```
from matplotlib.rcsetup import validate_font_properties
model.compile(optimizer = 'adam',
              loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits =␣
 ↪True),
              metrics = ['accuracy'])

# numero de epocas podemos jugar para mejorar la presicion
history = model.fit(train_images, train_labels, epochs = 25, validation_data =␣
 ↪(test_images, test_labels))
```

```
Epoch 1/25
1875/1875 [==============================] - 11s 5ms/step - loss: 0.4517 -
accuracy: 0.8351 - val_loss: 0.3534 - val_accuracy: 0.8739
Epoch 2/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2966 -
accuracy: 0.8916 - val_loss: 0.3222 - val_accuracy: 0.8837
Epoch 3/25
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2494 -
accuracy: 0.9078 - val_loss: 0.2600 - val_accuracy: 0.9063
Epoch 4/25
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2177 -
accuracy: 0.9197 - val_loss: 0.2708 - val_accuracy: 0.9056
Epoch 5/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1894 -
```

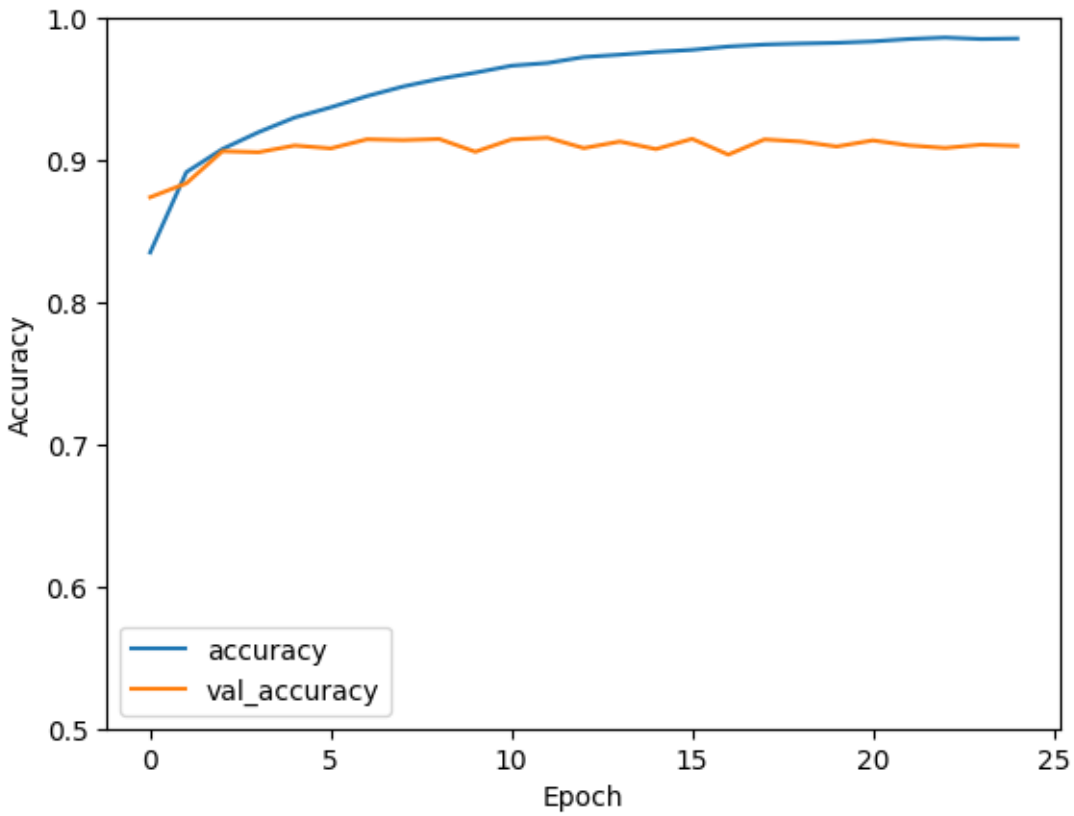
```
accuracy: 0.9300 - val_loss: 0.2515 - val_accuracy: 0.9103
Epoch 6/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1668 -
accuracy: 0.9372 - val_loss: 0.2746 - val_accuracy: 0.9083
Epoch 7/25
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1478 -
accuracy: 0.9451 - val_loss: 0.2577 - val_accuracy: 0.9147
Epoch 8/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1287 -
accuracy: 0.9518 - val_loss: 0.2705 - val_accuracy: 0.9141
Epoch 9/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1123 -
accuracy: 0.9572 - val_loss: 0.2844 - val_accuracy: 0.9149
Epoch 10/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1017 -
accuracy: 0.9616 - val_loss: 0.3253 - val_accuracy: 0.9059
Epoch 11/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0890 -
accuracy: 0.9665 - val_loss: 0.3254 - val_accuracy: 0.9146
Epoch 12/25
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0833 -
accuracy: 0.9684 - val_loss: 0.3592 - val_accuracy: 0.9158
Epoch 13/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0732 -
accuracy: 0.9726 - val_loss: 0.3460 - val_accuracy: 0.9085
Epoch 14/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0676 -
accuracy: 0.9742 - val_loss: 0.4117 - val_accuracy: 0.9131
Epoch 15/25
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0629 -
accuracy: 0.9762 - val_loss: 0.3978 - val_accuracy: 0.9078
Epoch 16/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0614 -
accuracy: 0.9776 - val_loss: 0.4097 - val_accuracy: 0.9150
Epoch 17/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0524 -
accuracy: 0.9800 - val_loss: 0.4951 - val_accuracy: 0.9039
Epoch 18/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0504 -
accuracy: 0.9814 - val_loss: 0.4714 - val_accuracy: 0.9146
Epoch 19/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0480 -
accuracy: 0.9821 - val_loss: 0.4636 - val_accuracy: 0.9131
Epoch 20/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0471 -
accuracy: 0.9826 - val_loss: 0.4922 - val_accuracy: 0.9096
Epoch 21/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0447 -
```

```
accuracy: 0.9836 - val_loss: 0.5087 - val_accuracy: 0.9139
Epoch 22/25
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0385 -
accuracy: 0.9852 - val_loss: 0.5210 - val_accuracy: 0.9104
Epoch 23/25
1875/1875 [==============================] - 8s 5ms/step - loss: 0.0382 -
accuracy: 0.9863 - val_loss: 0.5558 - val_accuracy: 0.9086
Epoch 24/25
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0419 -
accuracy: 0.9852 - val_loss: 0.5701 - val_accuracy: 0.9109
Epoch 25/25
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0404 -
accuracy: 0.9856 - val_loss: 0.5659 - val_accuracy: 0.9100
```

2.6 Evaluación

[97]:
```
plt.plot(history.history['accuracy'],label = 'accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.ylim([0.5,1])
```

[97]: (0.5, 1.0)



[98]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.5659 - accuracy: 0.9100 - 718ms/epoch - 2ms/step
```

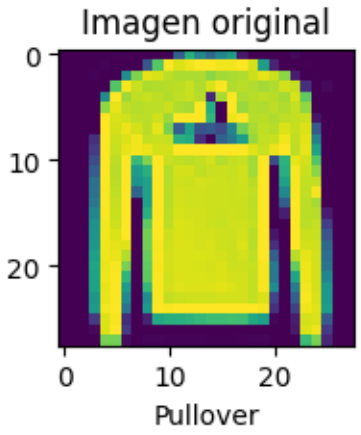
[99]:
```
print("Accuracy: ", test_acc)
```

```
Accuracy:  0.9100000262260437
```

2.7 Predecir

dentro de model despues del fit ahora estan los pesos y las estructuras

[100]:
```
n= 115 #numero de imagen (total de 60000 imagenes)

plt.figure(figsize= (2,2))
plt.imshow(test_images[n])
plt.xlabel(class_names[test_labels[n]])
plt.title('Imagen original')
plt.show()
```



[101]:
```
predictions = model.predict(test_images)
print(predictions[n]) #probabilidades de cada una de las neruonas en la ultima␣
↪capa
```

```
313/313 [==============================] - 1s 2ms/step
[5.4112636e-04 4.6971396e-14 1.0000000e+00 6.4067180e-12 2.0459828e-01
 1.3794807e-03 6.1761029e-03 8.0866054e-13 7.7909426e-14 9.6753413e-12]
```

[102]:
```
import numpy as np
print("La imagen pertence al grupo {} con una probabilidad de {:.2f} %"
```

```
.format(class_names[np.argmax(predictions[n])], 100 + np.max(predictions[n])))
```

```
La imagen pertence al grupo Pullover con una probabilidad de 101.00 %
```

3 Conclusión

Para el ejercicio de Cifar10, se sabe que son imagenes en RGB por ende desde un principio se sabe que el input es de 3 canales; también se hacen 3 capas de convolución comenzando con 64 hasta alcanzar las 256 neuronas; y al momento de hacer las capas densas, y observando la arquitectura del modelo de estar la capa (4x4x256) se hace un Flatten para que se convierta a 1 dimensión con output de (None,10). Continuando con el entrenamiento, se realizaron varias pruebas, ya que a pesar de que el numero de neuronas influye en el accuracy, también lo hace el número de épocas, por ende, para evitar un overfitting se realizaron 5 epocas en el cual, se logra alcanzar el accuracy por encima del 70%, el cual en mi ejercicio fue del 72%; teniendo así una predicción donde la imagen dada como input fue un Cat y la predicción fue un Cat.

Hablando sobre Fashion_mnist, se toma en cuenta que las imagenes de este dataset, no vienen en un formato de RGB sino en grayscale, por ende, en vez de 3 canales como en Cifar10, aquí es 1 canal, ya que solo se obtiene 0 ó 1 como valores de entrada, este "1" fue declarado en el input de la convulación, donde se observa que su arquitectura llega a (3x3x128) y gracias a las capas densas llega a (None, 10). Dentro del entrenamiento se realizan 25 épocas ya que jugando un poco con este número se obtiene el mejor accuracy con un 91%, el cual al momento de hacer la predicción se pone como input una imagen "Pullover", y la predicción es correcta obteniendo "Pullover"