

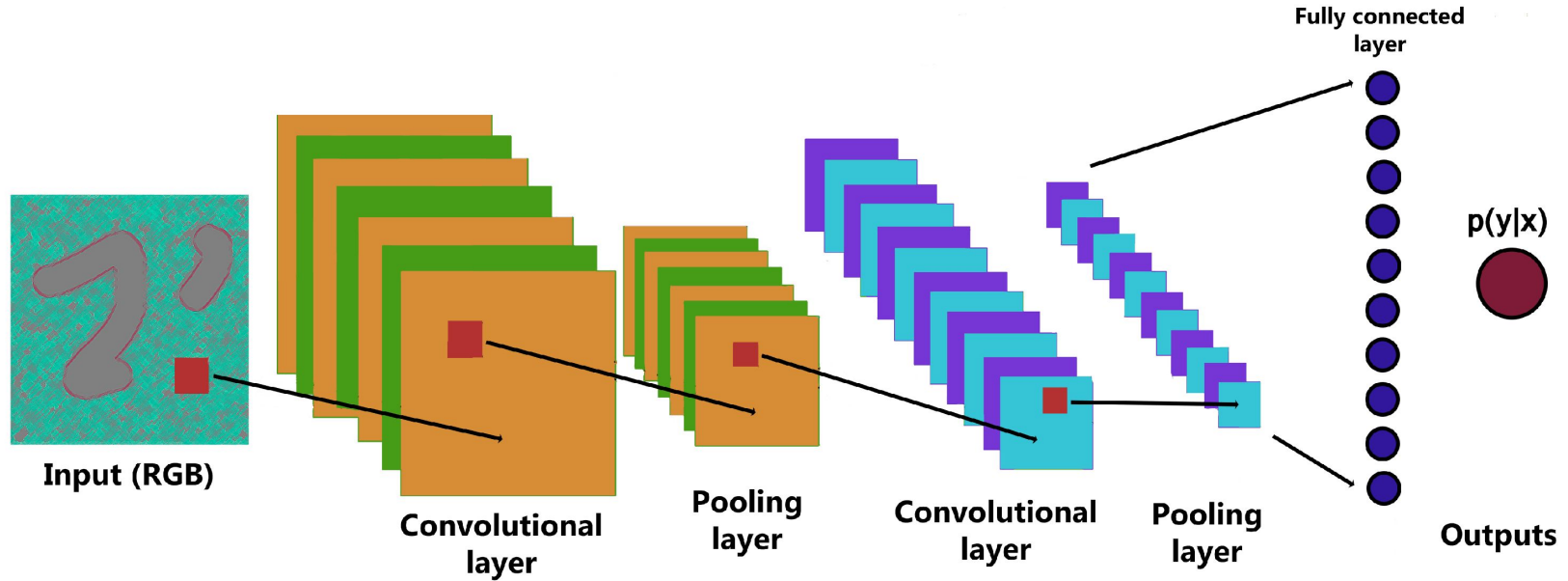
# Modelo de red neuronal reducido para la segmentación de grietas en el pavimento

M. en C. José Uriel González Escalona

# Contenido del taller

- Redes neuronales convolucionales
- Segmentación binaria
- Grietas en el pavimento
- U-Net
- Redes reducidas
- Operador Depth-to-space
- Redes neuronales profundas con el operador Depth-to-space para la segmentación de grietas en el pavimento
- Medida de desempeño – F1-score

# Red neuronales convolucionales



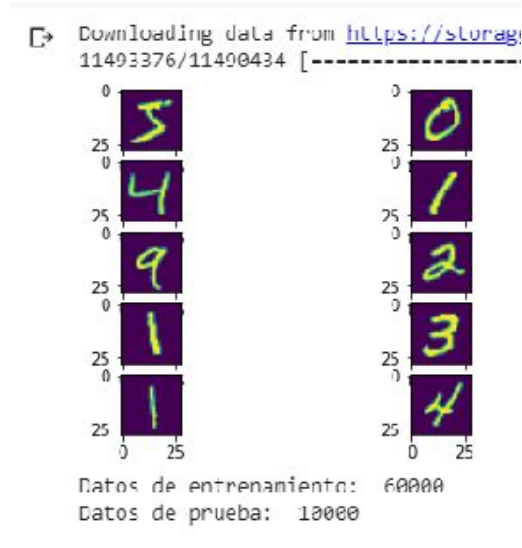
**Reto:** Completar el código para el entrenamiento de una red neuronal convolucionaria utilizando la base de datos MNIST

- Cargar la base de datos de Keras
- Utilizar la distribución de entrenamiento y validación inicial
- Probar los resultados en 10 números del set de prueba

**Tips:** Buscar las librerías:

- Numpy
- Matplotlib
- Keras

## Resultado Esperado:



RIIAA-1

## Solución:

```
[1] #C1
# Importar librerías

import numpy as np
import matplotlib.pyplot as plt
import time

import keras

# Importar MNIST
from keras.datasets import mnist

# Uso de One-hot encoder
from keras.utils import to_categorical

# Librerías para la red convolucional
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
```

```
[2] # C2
# Entrenamiento de una red neuronal convolucional con MNIST

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Visualizar los datos
fig = plt.figure()
for i in range(0,10):
    a = fig.add_subplot(5,2,i+1)
    plt.imshow(x_train[i])
plt.show()

print('Datos de entrenamiento: ', len(x_train))
print('Datos de prueba: ', len(x_test))

# Reescalar los datos
x_train = x_train.reshape(60000,28,28,1)
x_test = x_test.reshape(10000,28,28,1)

# Codificación one-hot para las salidas
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
[4] #C3
# Entrenamiento de una red convolucional con MNIST

# Creación del modelo
model = Sequential() #Iniciar modelo secuencial

# Añadir capas a la red
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

```
[5] #C4
# Compilar el modelo y seleccionar las métricas

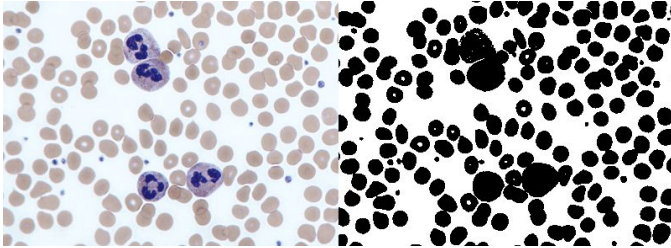
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[6] #C5
# Entrenar el modelo, selección de parámetros y número de épocas

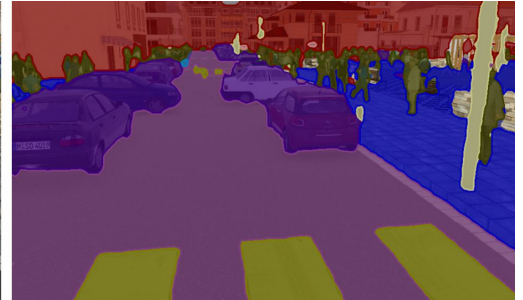
model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=30)
```

# Segmentacion binaria

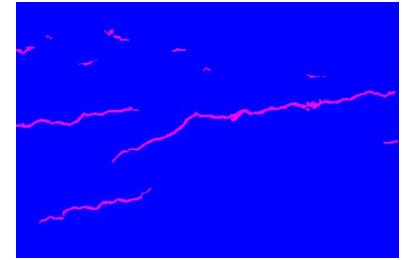
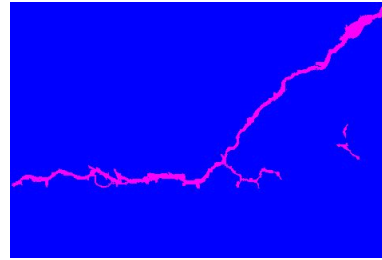
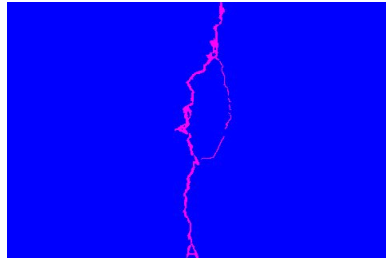
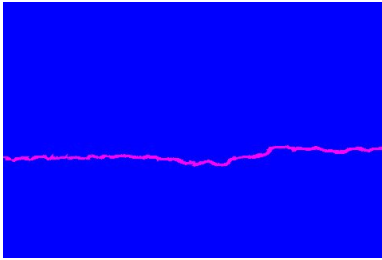
Binarización



Segmentación



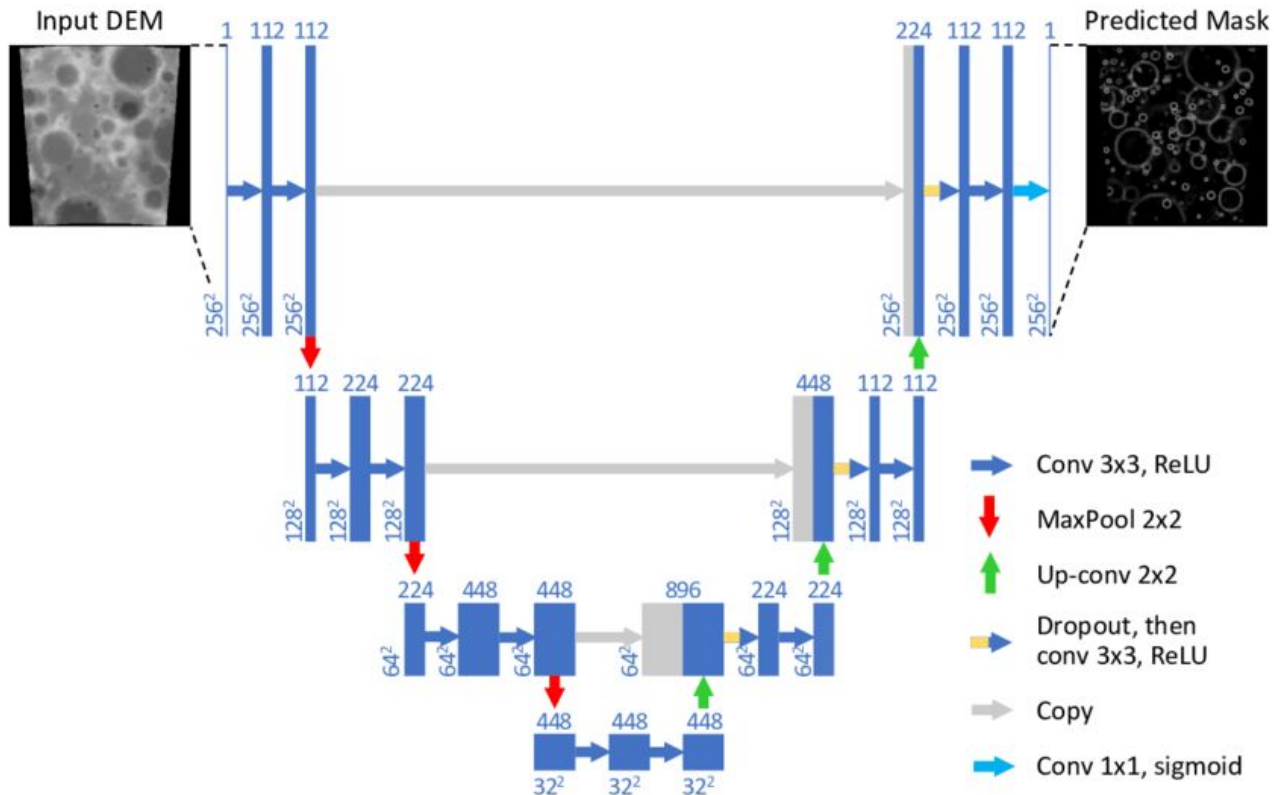
# Grietas en el pavimento



- Producidas por desgaste del suelo
- Carga pesada
- Falta de mantenimiento

- ✓ Segmentación manual
- ✓ Reducción el tamaño de las imágenes originales

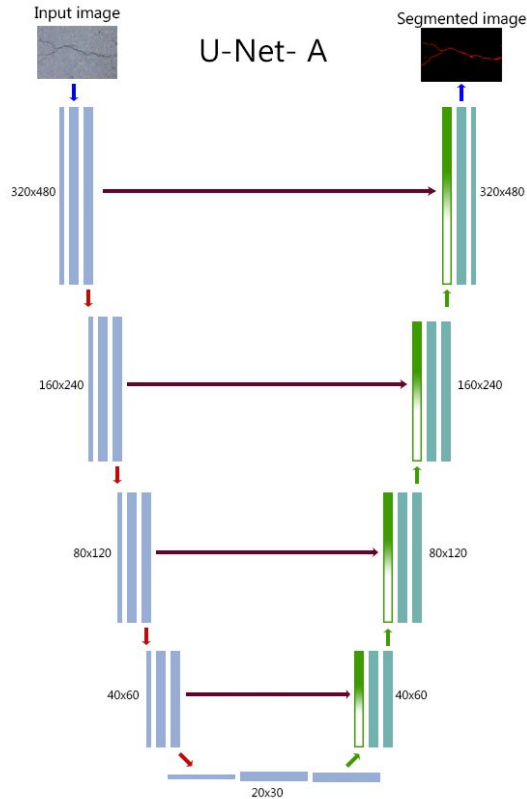
## Red neuronales convolucionales – U-Net



## U-Net



# U-Net



No. de parámetros:  
31,031,810

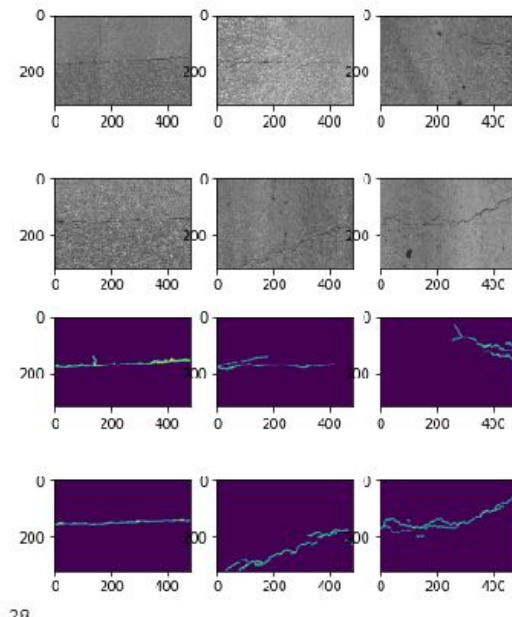
Layer (type)	Output Shape	Param #	Connected to
Input_1 (InputLayer)	(None, 320, 480, 3)	0	
conv2d_1 (Conv2D)	(None, 320, 480, 64)	1792	Input_1[0][0]
conv2d_2 (Conv2D)	(None, 320, 480, 64)	36928	conv2d_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 160, 240, 64)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 160, 240, 128)	73856	max_pooling2d_1[0][0]
conv2d_4 (Conv2D)	(None, 160, 240, 128)	147584	conv2d_3[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 80, 120, 128)	0	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 80, 120, 256)	295168	max_pooling2d_2[0][0]
conv2d_6 (Conv2D)	(None, 80, 120, 256)	590880	conv2d_5[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 40, 60, 256)	0	conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 40, 60, 512)	1180160	max_pooling2d_3[0][0]
conv2d_8 (Conv2D)	(None, 40, 60, 512)	2358880	conv2d_7[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 20, 30, 512)	0	conv2d_8[0][0]
conv2d_9 (Conv2D)	(None, 20, 30, 1024)	4719616	max_pooling2d_4[0][0]
conv2d_10 (Conv2D)	(None, 20, 30, 1024)	9438208	conv2d_9[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 40, 60, 1024)	0	conv2d_10[0][0]
conv2d_11 (Conv2D)	(None, 40, 60, 512)	2097664	up_sampling2d_1[0][0]
concatenate_1 (Concatenate)	(None, 40, 60, 1024)	0	conv2d_11[0][0] conv2d_10[0][0]
conv2d_12 (Conv2D)	(None, 40, 60, 512)	4719280	concatenate_1[0][0]
conv2d_13 (Conv2D)	(None, 40, 60, 512)	2358880	conv2d_12[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 80, 120, 512)	0	conv2d_13[0][0]
conv2d_14 (Conv2D)	(None, 80, 120, 256)	524544	up_sampling2d_2[0][0]
concatenate_2 (Concatenate)	(None, 80, 120, 512)	0	conv2d_14[0][0] conv2d_13[0][0]
conv2d_15 (Conv2D)	(None, 80, 120, 256)	1179984	concatenate_2[0][0]
conv2d_16 (Conv2D)	(None, 80, 120, 256)	590880	conv2d_15[0][0]
up_sampling2d_3 (UpSampling2D)	(None, 160, 240, 256)	0	conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 160, 240, 128)	131280	up_sampling2d_3[0][0]
concatenate_3 (Concatenate)	(None, 160, 240, 256)	0	conv2d_17[0][0] conv2d_16[0][0]
conv2d_18 (Conv2D)	(None, 160, 240, 128)	295040	concatenate_3[0][0]
conv2d_19 (Conv2D)	(None, 160, 240, 128)	147584	conv2d_18[0][0]
up_sampling2d_4 (UpSampling2D)	(None, 320, 480, 128)	0	conv2d_19[0][0]
conv2d_20 (Conv2D)	(None, 320, 480, 64)	32832	up_sampling2d_4[0][0]
concatenate_4 (Concatenate)	(None, 320, 480, 128)	0	conv2d_20[0][0] conv2d_19[0][0]
conv2d_21 (Conv2D)	(None, 320, 480, 64)	73792	concatenate_4[0][0]
conv2d_22 (Conv2D)	(None, 320, 480, 64)	36928	conv2d_21[0][0]
conv2d_23 (Conv2D)	(None, 320, 480, 2)	130	conv2d_22[0][0]
Total params: 31,031,810			
Trainable params: 31,031,810			
Non-trainable params: 0			

**Reto:** Complete el código para el entrenamiento de una red del tipo U-net para la segmentación de grietas en el pavimento.

- Utilice la base de datos AigleRN
- 75% para entrenamiento
- 25% para prueba
- Observe el comportamiento en imágenes de prueba

**Tips:** Observe como cargar la base de datos desde Google Drive

## Resultado Esperado:



RIIAA-2

## Solución:

```
#C1
# Importar librerías

import numpy as np
import matplotlib.pyplot as plt
import time

import keras

# Librerías para la red convolucional
from keras.models import *
from keras.layers import *
from keras.optimizers import *

# Librerías para cargar imágenes

from PIL import Image
import glob
```

```
[4] # C2
# Cargar la base de datos

# AigleRN - 38 imágenes

from google.colab import drive
drive.mount('/content/drive')

#Cargar la base de datos, directorio dependiente de cada persona
dir_img = '/content/drive/My Drive/Bases_de_datos/AigleRN/images'
dir_lab = '/content/drive/My Drive/Bases_de_datos/AigleRN/labels'

image_list = []
label_list = []

background_color = np.array([255, 0, 255]) ## Solo el magenta

for filename in glob.glob('/content/drive/My Drive/Bases_de_datos/AigleRN/images/*.png'):
    im=Image.open(filename)
    image_list.append(np.asarray(im))
    #image_list.append(np.asarray(np.expand_dims(im, axis=0)))

    filename1 = filename.replace('images', 'labels')
    filename2 = filename1.replace('unn', 'unn_road')
    la=Image.open(filename2)

    gt_image = np.asarray(la)
    gt_bg = np.all(gt_image == background_color, axis=2) #Binarizar la segmentación del target
    label_list.append(np.expand_dims(gt_bg,axis=2))

    #label_list.append(np.asarray(np.expand_dims(la, axis=0)))

print ('Total de imágenes: ', len(image_list))
print ('Total de labels: ', len(label_list))
```

# Solución:

```
[5] # C3
# Visualizar y acomodar los datos

# Visualizar los datos
fig = plt.figure()
for i in range(0,6):
    a = fig.add_subplot(2,3,i+1)
    plt.imshow(image_list[i])
    plt.show()

fig1 = plt.figure()
for j in range(0,6):
    b = fig1.add_subplot(2,3,j+1)
    plt.imshow(label_list[j][:,:0])
    plt.show()

# Separar los datos en conjunto de entrenamiento y prueba

from sklearn.model_selection import train_test_split

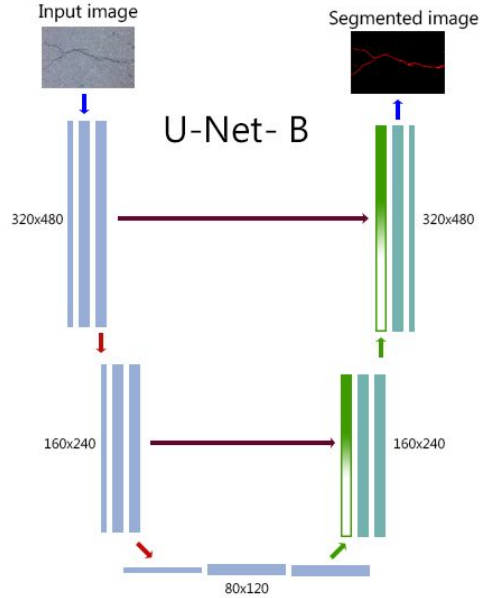
X_train, X_test, y_train, y_test = train_test_split(image_list, label_list, test_size=0.25, random_state=42) # Se requiere el 75% para entrenamiento

print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))
```

```
# C5
# Entrenar el modelo

model.fit(x=np.array(X_train)
          ,y=np.array(y_train)
          ,epochs=15 #Numero de epocas de entrenamiento
          ,batch_size=2
          ,validation_data=(np.array(X_test),np.array(y_test)))
```

# Redes reducidas



Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 320, 480, 3)	0	
conv2d_24 (Conv2D)	(None, 320, 480, 64)	1792	input_2[0][0]
conv2d_25 (Conv2D)	(None, 320, 480, 64)	36928	conv2d_24[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 160, 240, 64)	0	conv2d_25[0][0]
conv2d_26 (Conv2D)	(None, 160, 240, 128)	73856	max_pooling2d_5[0][0]
conv2d_27 (Conv2D)	(None, 160, 240, 128)	147584	conv2d_26[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, 80, 120, 128)	0	conv2d_27[0][0]
conv2d_28 (Conv2D)	(None, 80, 120, 256)	295168	max_pooling2d_6[0][0]
conv2d_29 (Conv2D)	(None, 80, 120, 256)	590880	conv2d_28[0][0]
up_sampling2d_5 (UpSampling2D)	(None, 160, 240, 256)	0	conv2d_29[0][0]
conv2d_30 (Conv2D)	(None, 160, 240, 128)	131200	up_sampling2d_5[0][0]
concatenate_5 (Concatenate)	(None, 160, 240, 256)	0	conv2d_30[0][0] conv2d_27[0][0]
conv2d_31 (Conv2D)	(None, 160, 240, 128)	295040	concatenate_5[0][0]
conv2d_32 (Conv2D)	(None, 160, 240, 128)	147584	conv2d_31[0][0]
up_sampling2d_6 (UpSampling2D)	(None, 320, 480, 128)	0	conv2d_32[0][0]
conv2d_33 (Conv2D)	(None, 320, 480, 64)	32832	up_sampling2d_6[0][0]
concatenate_6 (Concatenate)	(None, 320, 480, 128)	0	conv2d_33[0][0] conv2d_25[0][0]
conv2d_34 (Conv2D)	(None, 320, 480, 64)	73792	concatenate_6[0][0]
conv2d_35 (Conv2D)	(None, 320, 480, 64)	36928	conv2d_34[0][0]
conv2d_36 (Conv2D)	(None, 320, 480, 2)	130	conv2d_35[0][0]
Total params: 1,862,914			
Trainable params: 1,862,914			
Non-trainable params: 0			

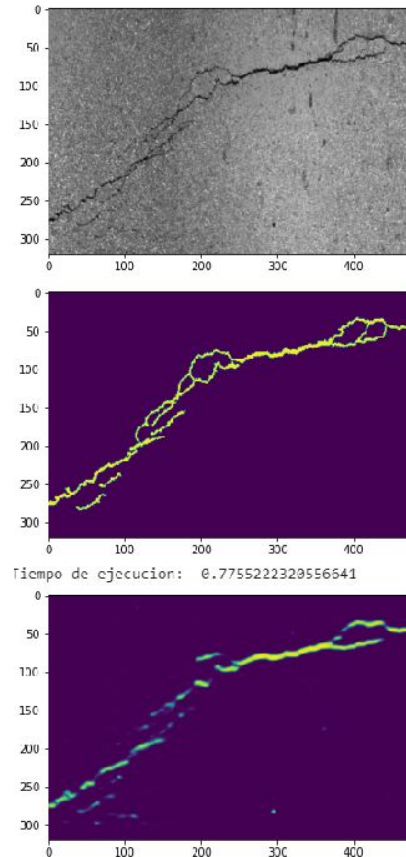
**Reto:** Completar el código para entrenar una red reducida utilizando imágenes de grieta en el pavimento

- Observe el número de parámetros de esta red
- Comente que diferencias tiene con la red U-Net

**Tips:** Observe el código del reto anterior

RIIAA-3

## Resultado Esperado:



## Solución:

```
#C1
# Importar librerías

import numpy as np
import matplotlib.pyplot as plt
import time

import keras

# Librerías para la red convolucional
from keras.models import *
from keras.layers import *
from keras.optimizers import *

# Librerías para cargar imágenes

from PIL import Image
import glob
```

```
[4] # C2
# Cargar la base de datos

# AigleRN - 38 imágenes

from google.colab import drive
drive.mount('/content/drive')

#Cargar la base de datos, directorio dependiente de cada persona
dir_img = '/content/drive/My Drive/Bases_de_datos/AigleRN/images'
dir_lab = '/content/drive/My Drive/Bases_de_datos/AigleRN/labels'

image_list = []
label_list = []

background_color = np.array([255, 0, 255]) ## Solo el magenta

for filename in glob.glob('/content/drive/My Drive/Bases_de_datos/AigleRN/images/*.png'):
    im=Image.open(filename)
    image_list.append(np.asarray(im))
    #image_list.append(np.asarray(np.expand_dims(im, axis=0)))

    filename1 = filename.replace('images', 'labels')
    filename2 = filename1.replace('um', 'um_road')
    la=Image.open(filename2)

    gt_image = np.asarray(la)
    gt_bg = np.all(gt_image == background_color, axis=2) #Binarizar la segmentación del target
    label_list.append(np.expand_dims(gt_bg,axis=2))

    #label_list.append(np.asarray(np.expand_dims(la, axis=0)))

print ('Total de imágenes: ', len(image_list))
print ('Total de labels: ', len(label_list))
```



## Solución:

```
[5] # C3
# Visualizar y acomodar los datos

# Visualizar los datos
fig = plt.figure()
for i in range(0,6):
    a = fig.add_subplot(2,3,i+1)
    plt.imshow(image_list[i])
    plt.show()

fig1 = plt.figure()
for j in range(0,6):
    b = fig1.add_subplot(2,3,j+1)
    plt.imshow(label_list[j][:,:0])
    plt.show()

# Separar los datos en conjunto de entrenamiento y prueba

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(image_list, label_list, test_size=0.25, random_state=42) # Se requiere el 75% para entrenamiento

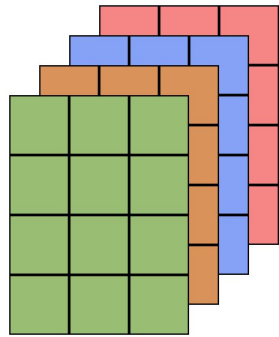
print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))
```

```
# C5
# Entrenar el modelo

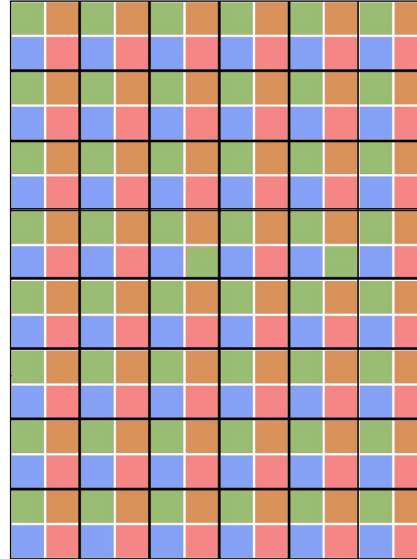
model.fit(x=np.array(X_train)
          ,y=np.array(y_train)
          ,epochs=15 #Numero de epocas de entrenamiento
          ,batch_size=2
          ,validation_data=(np.array(X_test),np.array(y_test)))
```



# Operador Depth-to-space



$4 \times 3 \times 1*2*2$



$2*4 \times 2*3 \times 1$

- Método rápido
- Un solo operador
- Obtener el tamaño original sin uso de deconvolución

$$(w, h, r^n) \Rightarrow (w * n, h * n, r)$$

$$n = 2 * \text{entero}$$

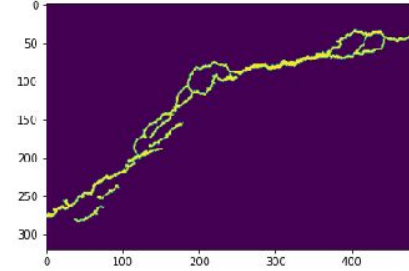
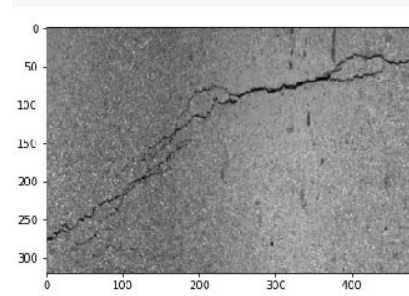
**Reto:** Completar el código para entrenar una red reducida utilizando el operador Depth-to-space

- Observe como se complementa el código anterior con este operador
- Mencione el número correcto del multiplicador de largo y ancho

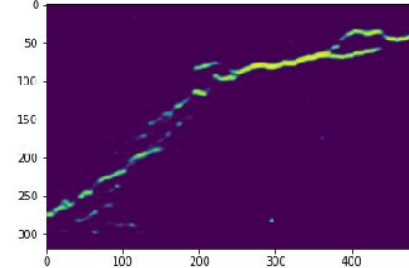
**Tips:** Observe el código del reto anterior

RIIAA-4

## Resultado Esperado:



Tiempo de ejecución: 0.7755222320556541



## Solución:

```
#C1
# Importar librerías

import numpy as np
import matplotlib.pyplot as plt
import time

import keras

# Librerías para la red convolucional
from keras.models import *
from keras.layers import *
from keras.optimizers import *

# Librerías para cargar imágenes

from PIL import Image
import glob

# Librería para Depth-to-space

import tensorflow as tf
```

```
# Visualizar y acomodar los datos

# Visualizar los datos
fig = plt.figure()
for i in range(0,6):
    a = fig.add_subplot(2,3,i+1)
    plt.imshow(image_list[i])
plt.show()

fig1 = plt.figure()
for j in range(0,6):
    b = fig1.add_subplot(2,3,j+1)
    plt.imshow(label_list[j][:,:0])
plt.show()

# Separar los datos en conjunto de entrenamiento y prueba

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(image_list, label_list, test_size=0.25, random_state=42)

print('Tamaño del set de entrenamiento: ', len(X_train))
print('Tamaño del set de prueba: ', len(X_test))
print(len(y_train))
print(len(y_test))
```

# Solución:

```
[ ] # Modelo Depth-to-space
```

```
def SubpixelConv2D(input_shape, scale=4):  
    def subpixel_shape(input_shape):  
        dims = [input_shape[0],  
                 input_shape[1] * scale,  
                 input_shape[2] * scale,  
                 int(input_shape[3] / (scale ** 2))]  
        output_shape = tuple(dims)  
        return output_shape  
  
    def subpixel(x):  
        return tf.nn.depth_to_space(x, scale)  
  
    return Lambda(subpixel, output_shape=subpixel_shape, name='subpixel')
```

```
[ ] # Modelo: Reducido con Depth-to-space
```

```
input_size = (320,480,3)
```

```
size = (5,5)
```

```
def unetll(input_size = (320,480,3)):
```

```
    inputs = Input(input_size)
```

```
    s = Lambda(lambda x: x / 255)(inputs)
```

```
    conv1 = Conv2D(64, size, padding='same', activation='relu')(s)
```

```
    conv2 = Conv2D(64, size, padding='same', activation='relu')(conv1)
```

```
    pool1 = AveragePooling2D(pool_size=(2, 2))(conv2)
```

```
    conv3 = Conv2D(128, size, padding='same', activation='relu')(pool1)
```

```
    conv4 = Conv2D(128, size, padding='same', activation='relu')(conv3)
```

```
    pool2 = AveragePooling2D(pool_size=(2, 2))(conv4)
```

```
    conv5 = Conv2D(256, size, padding='same', activation='relu')(pool2)
```

```
    conv6 = Conv2D(256, size, padding='same', activation='relu')(conv5)
```

```
    sub1 = SubpixelConv2D(conv6.shape, scale = 4)(conv6)
```

```
    conv7 = Conv2D(64, size, padding='same', activation='relu')(sub1)
```

```
    conv8 = Conv2D(64, size, padding='same', activation='relu')(conv7)
```

```
    merge1 = concatenate([conv8, conv2], axis=3)
```

```
    conv9 = Conv2D(128, size, padding='same', activation='relu')(merge1)
```

```
    y = Conv2D(1, (1, 1), activation='sigmoid')(conv9)
```

```
    return Model(inputs=inputs, outputs=y)
```

```
model = unetll(input_size)
```

```
model.compile(optimizer = Adam(lr = 1e-4), loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
model.summary()
```

# Medida de desempeño

## F1-score

$$Precision = \frac{T_p}{T_p + F_p}$$

$$Recall = \frac{T_p}{T_p + F_n}$$

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

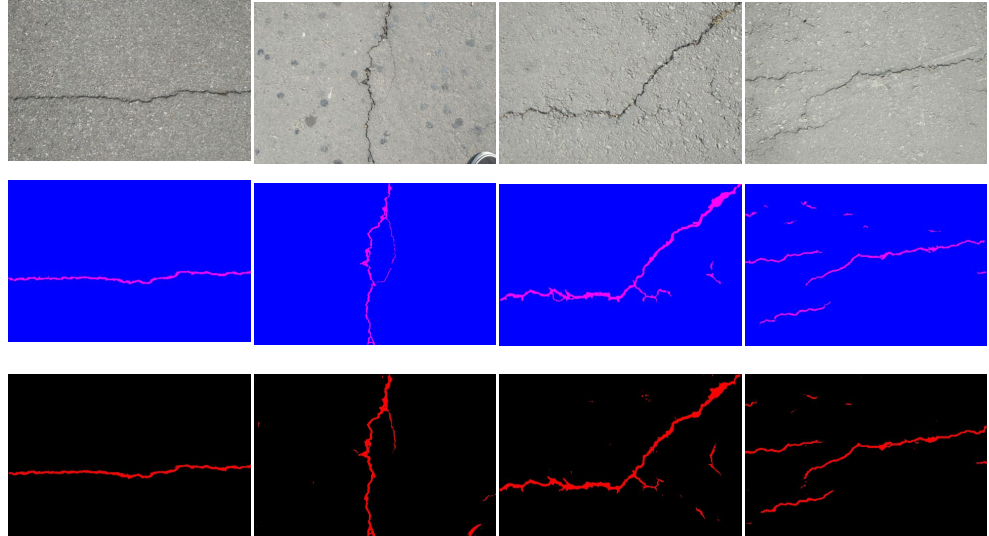
Valor entre 0 y 1

$T_p$  = Verdadero positivo

$T_n$  = Verdadero negativo

$F_p$  = Falso positivo

$F_n$  = Falso negativo



## Solución:

```
# Probar el modelo con una sola imagen

plt.imshow(X_test[0])
plt.show()

t_l = y_test [0]
t1 = t_l[:, :, 0]

plt.imshow(t1)
plt.show()

inicio = time.time()
pred = model.predict(np.expand_dims(X_test[0], axis=0))
fin = time.time()
print ('Tiempo de ejecucion: ', fin - inicio)

plt.imshow(pred[0, :, :, 0])
plt.show()
```

```
from keras import backend as K

# Segmentación manual
t_l = y_test [0]
t1 = t_l[:, :, 0]

# Convertirlo a digito
t1 = t1 * 1

# Predicción

predic = model.predict(np.expand_dims(X_test[0], axis=0))
pred = predic[0, :, :, 0]
# Aplicar un umbral de decisión
threshold = 0.01
predh = [pred > threshold]
predh = np.multiply(predh, 1)
predh = predh[0, :, :]

# Para poder encontrar el handicap de los 2 pixeles se utiliza la siguiente funcion
tp = 0
fp = 0
fn = 0
h = 2 ##### numero de pixeles alejado del punto a revisar
for r in range(len(t1)):
    for o in range(len(t1[0])):
        if t1[r][o] == 1 and predh[r-h:r+h, o-h:o+h].sum() >= 1:
            tp += 1
        elif t1[r][o] == 1 and predh[r][o] == 0:
            fp += 1
        elif t1[r][o] == 0 and predh[r][o] == 1:
            fn += 1
pr = tp/(tp+fp +K.epsilon())
re = tp/(tp+fn +K.epsilon())
f1 = 2*(pr*re)/(pr+re +K.epsilon())

print('El valor de precision es: ', pr)
print('El valor de recall es: ', re)
print('El valor de F1 - Score es: ', f1)
```

## Solución:

```
# Valor de F1-para todos los elementos del set de prueba

for i in range (0,10):
    # Segmentación manual
    t_l = y_test [i]
    t1 = t_l[:, :, 0]

    # Convertirlo a digito
    t1 = t1 * 1

    #Predicción

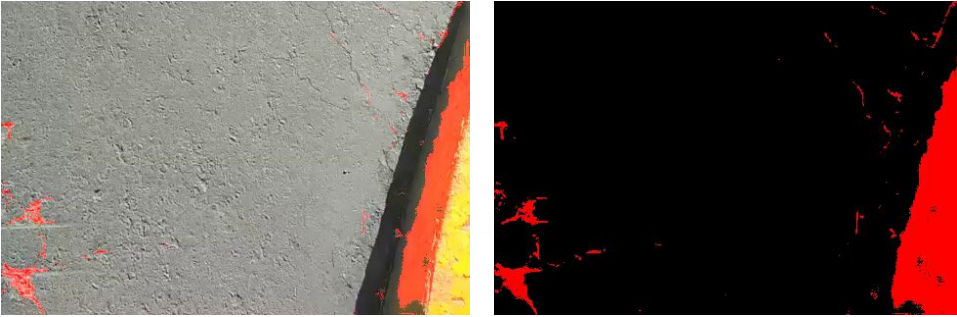
    predic = model.predict(np.expand_dims(X_test[i],axis=0))
    pred = predic[0, :, :, 0]
    # Aplicar un umbral de decisión
    threshold = 0.01
    predh = [pred > threshold]
    predh = np.multiply(predh,1)
    predh = predh[0, :, :]

    # Para poder encontrar el handicap de los 2 pixeles se utiliza la siguiente funcion
    tp = 0
    fp = 0
    fn = 0
    h = 2 ##### numero de pixeles alejado del punto a revisar
    for r in range(len(t1)):
        for c in range(len(t1[0])):
            if t1[r][c] == 1 and predh[r-h:r+h, c-h:c+h].sum() >= 1:
                tp += 1
            elif t1[r][c] == 1 and predh[r][c] == 0:
                fp += 1
            elif t1[r][c] == 0 and predh[r][c] == 1:
                fn += 1
    pr = tp/(tp+fp +K.epsilon())
    re = tp/(tp+fn +K.epsilon())
    f1 = 2*(pr*re)/(pr+re +K.epsilon())

    #print('El valor de precision es: ', pr)
    #print('El valor de recall es: ', re)
    print('El valor de F1 - score es: ', f1)
```



## Uso en ambientes reales



- Uso de la red en tiempo real
- Velocidad de procesado: 0.09s
- Hasta 12 fps

- Cambios de iluminación
- Vegetación
- Sombras
- Ruidos reales (basura, aceite, agua)



# ¡Gracias!

## Cursos



### Python para IA

Programa en python aplicaciones de IA

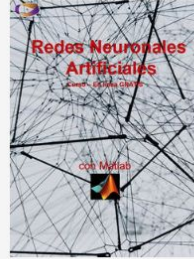
MÁS INFORMACIÓN



### PLN para IA

Haz que una maquina converse

MÁS INFORMACIÓN



### Redes Neuronales Artificiales

Aprende los conceptos básicos (GRATIS)

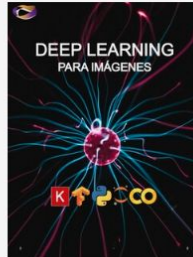
MÁS INFORMACIÓN



### Machine Learning Práctico

Programa algoritmos que aprendan de los datos

MÁS INFORMACIÓN



### Deep Learning para Imágenes

Haz que una maquina reconozca imágenes

MÁS INFORMACIÓN



### Visión por Computadora con Python

Aprende a procesar imágenes (GRATIS)

MÁS INFORMACIÓN

<https://www.actumlogos.com/>

[CURSOS](#)

[GRATIS](#)

[EQUIPO](#)

[CONTACTO](#)

[BLOG](#)

Participa en la revolución tecnológica



**ACTUMLOGOS**  
DESARROLLANDO HABILIDADES TECNOLÓGICAS

[hola@actumlogos.com](mailto:hola@actumlogos.com)

[wa.me/5215539940156](https://wa.me/5215539940156)

[Facebook](#)

[Linkedin](#)

# Referencias

<https://arxiv.org/abs/1505.04597> - U-Net: Convolutional Networks for Biomedical Image Segmentation

<https://arxiv.org/abs/1805.00138> - Semantic Binary Segmentation using Convolutional Networks without Decoders

<https://arxiv.org/abs/1609.05158> - Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network

<https://arxiv.org/abs/1707.05847> - The Devil is in the Decoder: Classification, Regression and GANs

<https://ieeexplore.ieee.org/document/7471507> - Automatic Road Crack Detection Using Random Structured Forests

<https://arxiv.org/abs/1802.02208> - Automatic Pavement Crack Detection Based on Structured Prediction with the Convolutional Neural Network

# Conclusión Final

- Una red neuronal puede trabajar con imágenes de una forma clara.
- Se puede clasificar (MNIST)
- Se puede segmentar (AigleRN)
- El tiempo de procesamiento de las redes neuronales convolucionales depende de su número de hiperparametros
- Es posible reducir el número de los hiperparametros sin disminuir su efectividad
- El operador Depth-to-space se puede utilizar para realizar una deconvolucion rápida
- La segmentación binaria es útil para resaltar problemas en el pavimento

**Reto:**

**Resultado Esperado:**

**Tips:**

Solución:

***¡Plus ultra!*** ...ir más allá

# Glosario