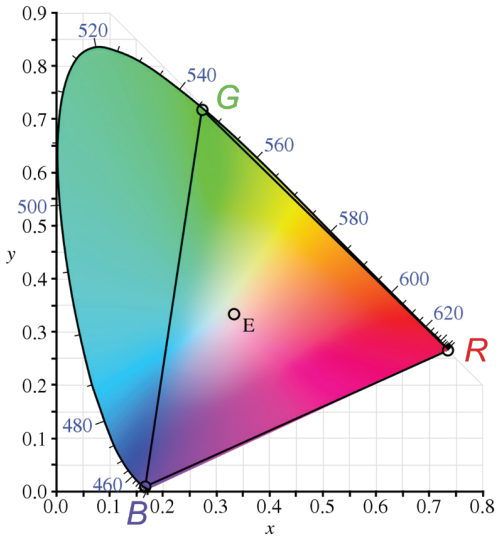


Figura 1.1: Diagrama cromático CIE 1931. Solo se visualizan correctamente los colores reproducibles a partir de las (pocas) tintas empleadas en la impresión de este documento. (Adaptado de Wikipedia, CIE 1931 color space).

En los años 30 se realizaron una serie de experimentos encaminados a relacionar cuantitativamente la radiación electromagnética con la percepción visual. Fruto de ello se propuso el denominado espacio color CIE 1931 XYZ, que asigna a cada color 3 números positivos, de forma unívoca. Estos números pueden entenderse como las coordenadas de cada color en un espacio cromático tridimensional denominado XYZ. Por conveniencia, el espacio XYZ fue definido de forma que Y representa la luminosidad, es decir la cantidad total de luz, mientras que X y Z se relacionan con el color propiamente dicho.

Los *diagramas cromáticos* son una forma adecuada de manejar este espacio cromático (véase la Figura 1.1). Se trata de secciones planas del espacio XYZ en las que se representa la gama de colores que puede percibir el observador humano promedio, para un determinado nivel de luminosidad (un cierto valor de Y). Los *colores espectrales*, es decir correspondientes a las radiaciones monocromáticas (de una única longitud de onda), se sitúan en el contorno de la gama, mientras que la denominada *línea de púrpuras* delimita la región por la parte inferior. Por construcción, todos los colores de la gama pueden generarse combinando luces de los (infinitos) colores espectrales, balanceadas en función del inverso de su distancia al color deseado medida sobre el diagrama. La característica forma de *lengua* o *herradura* de estos diagramas deriva de la fuerte correlación existente entre las respuestas espectrales de los distintos tipos de conos.

En general, un conjunto arbitrario de luces permite generar todos los colores presentes en el interior del polígono que circunscriben dentro del diagrama cromático (pero no los presentes en el exterior). A modo de ejemplo, el espacio

vector de salida se debe poner a 1 la clase esperada y a 0 el resto de clases para cada muestra. Esto hará que el modelo vaya aprendiendo poco a poco a aproximar el valor de salida a 1 para la categoría inferida. Es necesario tener en cuenta que, al ser el sumatorio de las probabilidades de salida igual a 1 para la función *softmax*, esta técnica no será la ideal para problemas de multi-clasificación, donde esperamos distintas clases de salida. Dichos problemas se pueden resolver principalmente con distintos modelos para cada tipo de categoría.

5.4.1. Entropía cruzada

Anteriormente se vio cómo en el ejemplo de la clasificación binaria de si llueve o no se había utilizado la entropía cruzada binaria. La entropía de una variable es el nivel de incertidumbre respecto a las posibles variables del resultado. En la teoría de la información, la entropía se calcula como se muestra en la Ecuación 5.16.

$$H(x) = -\sum_{x \in X} p(x) \cdot log(p(x)) \tag{5.16}$$

Supongamos que tenemos tres vasos con bolas de distintos colores, como se observa en la Figura 5.7. La entropía de cada uno de los vasos vendría determinada por las Ecuaciones 5.17 a 5.19. Podemos ver cómo el vaso 2 es el que tiene menor entropía, ya que la probabilidad de que alguna de las bolas en su interior sea naranja es mucho más elevada que el resto, por ejemplo de que sea verde en el vaso 3.

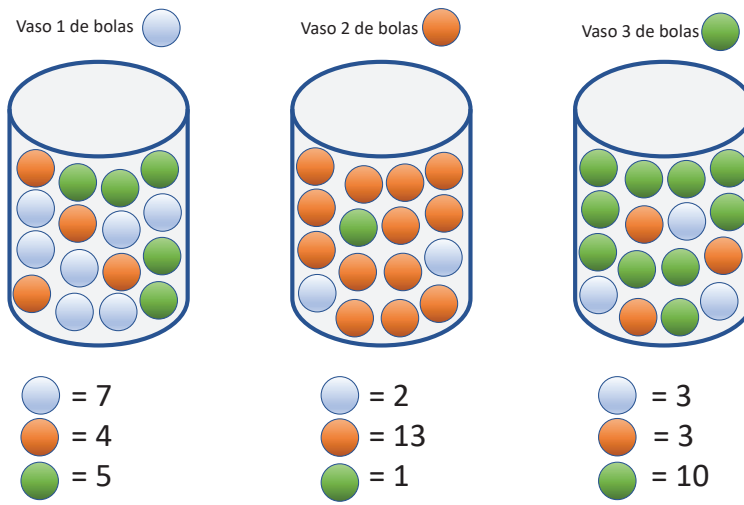


Figura 5.7: Vasos con bolas de distintos colores

```
# Lista de clases
clases = dataset.classes

# Posición de la imagen en el plot
posicion = 0
for c in range(len(clases)):
    # Buscamos las imágenes con esta clase
    indices = np.where(np.asarray(dataset.targets) == c)[0]

    # Escribimos etiqueta de clase
    posicion += 1
    plt.subplot(10, 11, posicion)
    plt.text(0, 0.5, clases[c], fontsize=6)
    plt.axis('off')

    # Sacamos 10 imágenes de cada categoría por fila
    for i in range(10):
        posicion += 1
        plt.subplot(10, 11, posicion)
        plt.imshow(dataset.data[indices[i]])
        plt.axis('off')
plt.show()
```

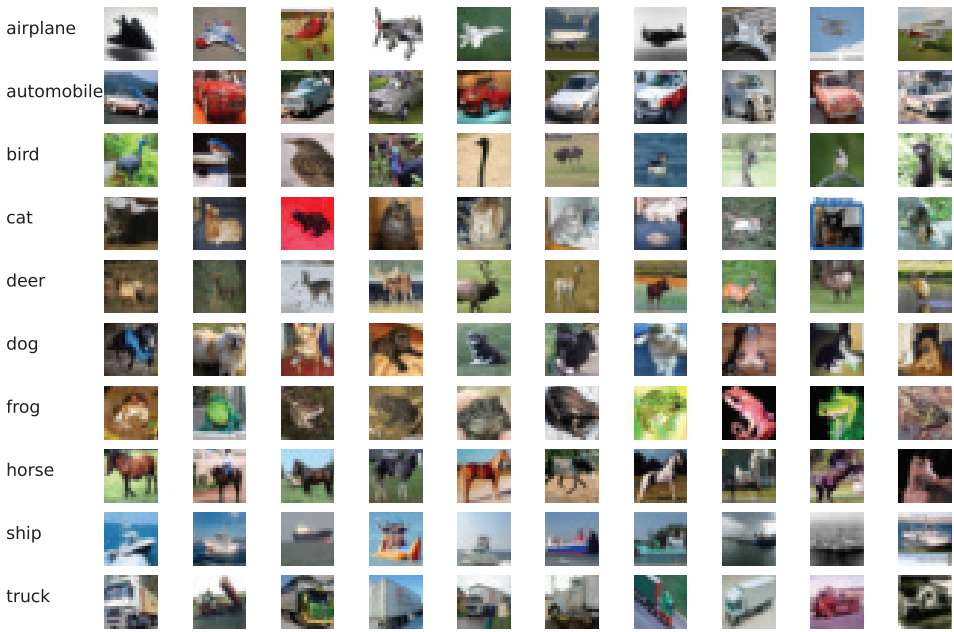


Figura 5.20: 100 imágenes de CIFAR-10 organizadas por categoría

En este ejemplo iremos profundizando en algunas técnicas más avanzadas del entrenamiento de una red neuronal convolucional. El dataset se puede utilizar

```
        jet_heatmap = array_to_img(jet_heatmap)
        jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0])
            )
        jet_heatmap = img_to_array(jet_heatmap)

        # Superponemos la imagen original y el heatmap
        img_superpuesta = jet_heatmap * alpha + img
        img_superpuesta = array_to_img(img_superpuesta)

        # Grabamos la imagen superpuesta
        img_superpuesta.save(out_path)

# Mostrar heatmap
plt.matshow(heatmap)
plt.show()
# Obtener y mostrar heatmap junto a imagen original
out_path = 'heatmap_conjunto_pavimento.png'
superponer_heatmap_imagen(PATH_FICHERO, heatmap, out_path)
# Mostramos la imagen superpuesta
display(Image(out_path))
```

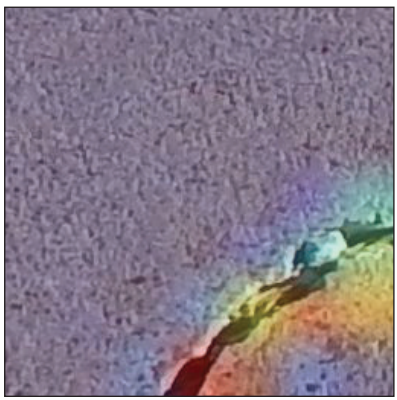
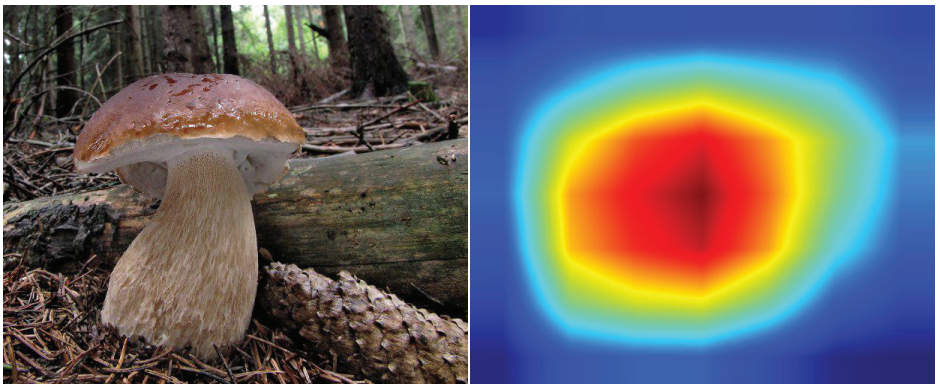


Figura 6.9: *Heatmap* final superpuesto con la imagen original del pavimento

Los *heatmaps* son una herramienta muy útil ya que nos permiten delimitar zonas de defectos, como en este caso. De hecho, a parte de los modelos de detección de objetos que veremos posteriormente, algunos trabajos sobre detección y segmentación de objetos en imágenes han partido de la utilización de *heatmaps*.

Finalmente, si queremos utilizar un modelo diferente a ResNet-50, como pueden ser las redes más modernas ResNeXt o ConvNeXt, habría que ver los modelos existentes dentro del módulo *tensorflow.keras.applications*. Normalmente, la mayoría de modelos se utilizan de una forma similar a ResNet-50, por lo que bastará con modificar el tipo de red en el encabezado de cara a la clasificación, y seleccionar las capas que queramos visualizar del nuevo tipo de red, a partir de un resumen que mostremos con la función *summary()*.

la parte convolucional (*weight_softmax*). Una vez realizada la inferencia sobre la imagen, pasaremos la clase ganadora, los pesos de enlace y los mapas de features de convolución, a la función previa *generarCAMs(...)* para obtener los mapas. Finalmente, guardaremos la imagen del *heatmap*, así como la imagen de la superposición de la imagen original con el mapa. La Figura 6.13 muestra la imagen original y el *heatmap* obtenido. Al igual que hacíamos con Tensorflow, utilizamos el mapa de colores *jet*, que es el más recomendable para este tipo de *heatmaps*.



(a) Imagen original (b) *Heatmap* obtenido

Figura 6.13: Imagen original y *heatmap* obtenido a partir de ResNeXt-101

En este caso, la predicción nos ha devuelto que la seta pertenecía a la clase "Boletus", que coincide con la imagen que habíamos seleccionado. En la Figura 6.14 podemos ver la superposición de la imagen original y el *heatmap* obtenido. Los valores de la imagen original se han ponderado por 0.5 mientras que los valores del *heatmap* se han ponderado por 0.3.

Figura 6.14: *Heatmap* final superpuesto con la imagen original de la seta

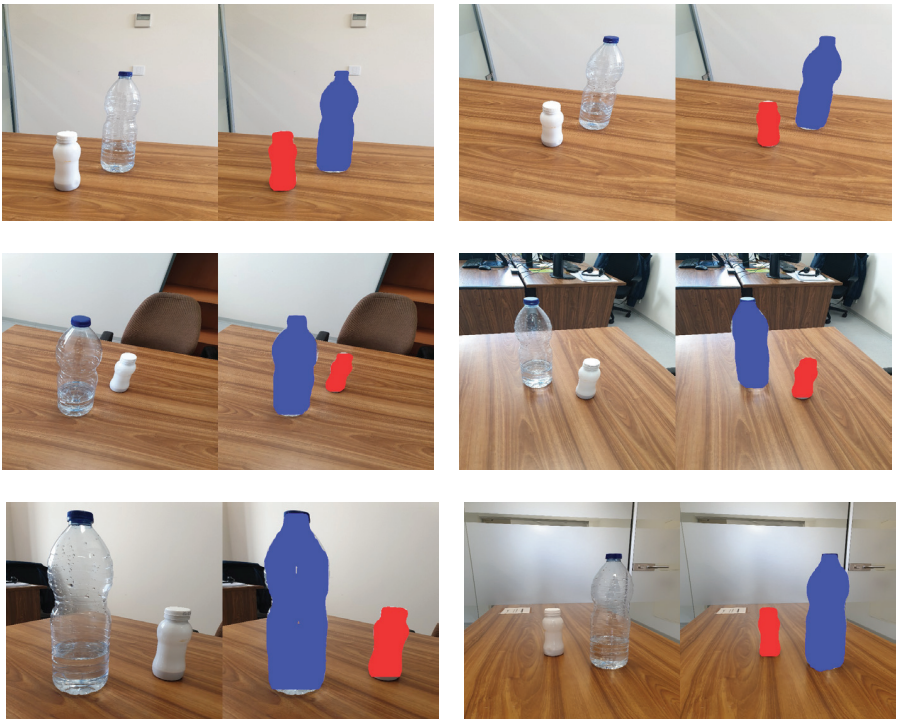


Figura 11.17: Resultados de la segmentación de algunas imágenes de test