Importaciones

In [2]:
```
import cv2
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np
import time
from IPython.display import display, Image

# Darkflow es una traducción de darknet a tensorflow pudiendo cargar los pesos entrei
from darkflow.net.build import TFNet
# Se requiere instalar darkflow del repositorio de github https://github.com/thtrieu/

# Sirve para que el backend de matplotlib renderice las figuras en formato 'svg'
%config InlineBackend.figure_format = 'svg'
```

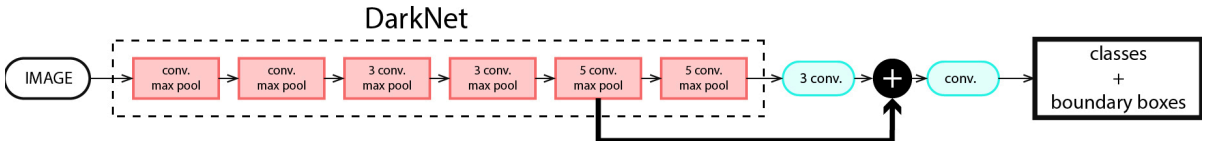
Cargamos el modelo

In [3]:
```
configuration = {
    # Especificamos el modelo y su ruta en mi caso es YoloV2
    'model': 'cfg/yolov2.cfg',
    # Cargamos los pesos
    'load': 'bin/yolov2.weights',
    # Como de buena debe de ser la predicción para considerarla válida
    'threshold': 0.3,
    # Los nombres de las etiquetas
    'labels': 'cfg/coco.names',
    # 1 Para usar la GPU (en mi caso) y 0 para tirar de CPU
    'gpu': 1
}
```

In [4]:
```
tfnet = TFNet(configuration)
```

```
Parsing ./cfg/yolov2.cfg
Parsing cfg/yolov2.cfg
Loading bin/yolov2.weights ...
Successfully identified 203934260 bytes
Finished in 0.020049571990966797s

Building net ...
```

```
Source | Train? | Layer description                | Output size
-------+--------+----------------------------------+---------------
WARNING:tensorflow:From C:\Users\alarc\anaconda3\envs\YooV2\lib\site-packages\tensorf
low\python\framework\op_def_library.py:263: colocate_with (from tensorflow.python.fra
mework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
       |        | input                            | (?, 608, 608, 3)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 608, 608, 32)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 304, 304, 32)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 304, 304, 64)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 152, 152, 64)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 152, 152, 128)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 152, 152, 64)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 152, 152, 128)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 76, 76, 128)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 76, 76, 256)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 76, 76, 128)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 76, 76, 256)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 38, 38, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 19, 19, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 19, 19, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 19, 19, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | concat [16]                      | (?, 38, 38, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 64)
 Load  |  Yep!  | local flatten 2x2                | (?, 19, 19, 256)
 Load  |  Yep!  | concat [27, 24]                  | (?, 19, 19, 1280)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 1x1p0_1    linear           | (?, 19, 19, 425)
-------+--------+----------------------------------+---------------
GPU mode with 1 usage
Finished in 7.248208999633789s
```



Procesar una imagen

Vamos a hacer una prueba con un objeto sencillo como lo es una taza

In [5]:
```
# cargamos la imagen, como al usar la función imread carga en BGR (Blue-Green-Red) p
img = cv2.cvtColor(cv2.imread('img/t.png', cv2.IMREAD_COLOR), cv2.COLOR_BGR2RGB)

# Devuelve <class 'numpy.ndarray'> de la forma (x,y,3) Siendo el 3 los canales de co
print(type(img))

result = tfnet.return_predict(img)
result
```

```
<class 'numpy.ndarray'>
```

Out[5]:
```
[{'label': 'cup',
  'confidence': 0.86425275,
  'topleft': {'x': 156, 'y': 20},
  'bottomright': {'x': 607, 'y': 469}}]
```

Podemos ver que el resultado al llamar a return_predict devuelve:

- Primero la **etiqueta** de clasificación
- La **confianza** de la predicción, es decir, como de seguro está
- Topleft y Bottonright sirven para especificar los puntos para crear la **caja delimitadora** de la predicción

[2]:

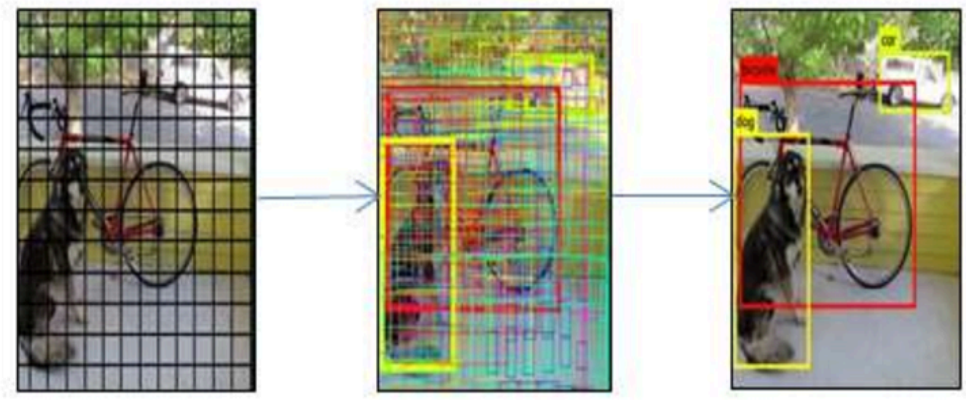


Fig. 1. YOLOv2 model regression.

Vamos a sacar ahora los valores y guardarlos como variables para usarlos en una imagen y cargar el resultado final

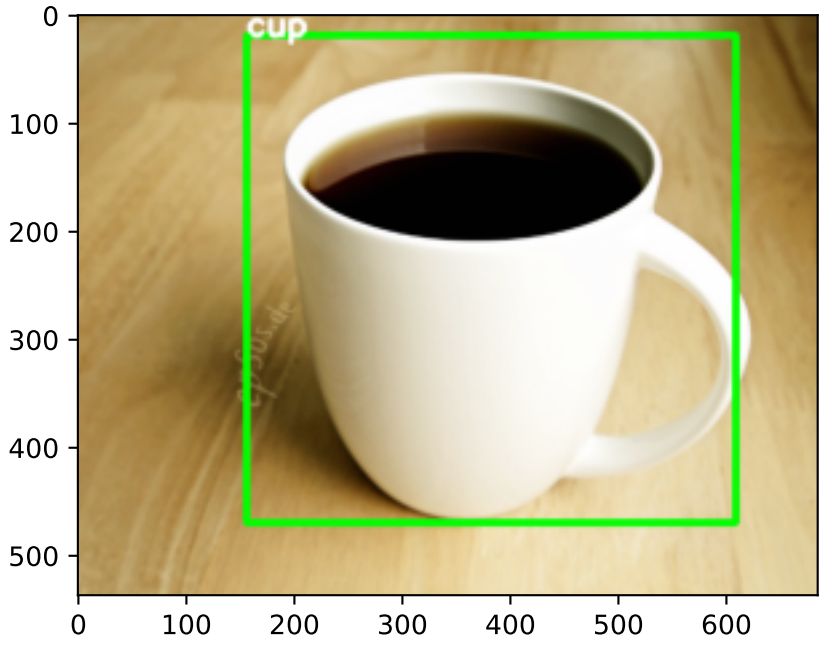
In [6]:
```
def valoresPrediccion(img):
    # cargamos la imagen, como al usar la función imread carga en BGR (Blue-Green-Re
    # Devuelve <class 'numpy.ndarray'> de la forma (x,y,3) Siendo el 3 los canales d
    img = cv2.cvtColor(cv2.imread('img/t.png', cv2.IMREAD_COLOR), cv2.COLOR_BGR2RGB)
    # Obtenemos la predicción
```

```
    result = tfnet.return_predict(img)

    label = result[0]['label']
    confidence = result[0]['confidence']
    topleft = result[0]['topleft']['x'],result[0]['topleft']['y']
    bottomright = result[0]['bottomright']['x'],result[0]['bottomright']['y']

    # Devuelvo los 4 valores
    return label,confidence,topleft,bottomright
```

In [7]:
```
label,confidence,topleft,bottomright= valoresPrediccion(result)
```

In [8]:
```
# Dibujamos sobre la imagen la caja
def showPrediction(img):
    img = cv2.rectangle(img, topleft, bottomright, (0,255,0), 5)
    # Podemos el texto de la predcción en la imagen
    img = cv2.putText(img, label, topleft,cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255)
    plt.imshow(img)
    plt.show()
```

In [9]:
```
showPrediction(img)
```



Pruebas con transformaciones básicas

In [10]:
```
def valoresPrediccionConImg(img_name):
    # cargamos la imagen, como al usar la función imread carga en BGR (Blue-Green-Re
    # Devuelve <class 'numpy.ndarray'> de la forma (x,y,3) Siendo el 3 los canales d
    img_path = f'img/{img_name}'

    img = cv2.cvtColor(cv2.imread(img_path, cv2.IMREAD_COLOR), cv2.COLOR_BGR2RGB)
    # Obtenemos la predicción
    result = tfnet.return_predict(img)

    label = result[0]['label']
    confidence = result[0]['confidence']
    topleft = result[0]['topleft']['x'],result[0]['topleft']['y']
```

```
    bottomright = result[0]['bottomright']['x'],result[0]['bottomright']['y']

    # Devuelvo los 4 valores
    return label,confidence,topleft,bottomright,img
```

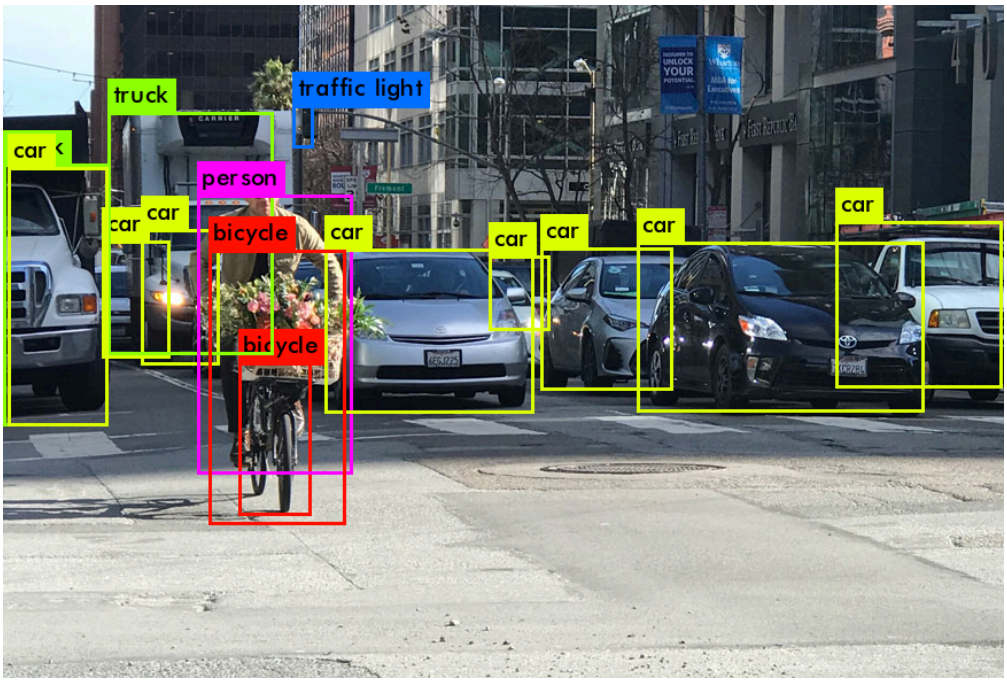
In [11]:

```
# Probamos con 4 variantes de la misma imagen
label,confidence,topleft,bottomright,img = valoresPrediccionConImg('t.png')
label3,confidence3,topleft3,bottomright3,img3 = valoresPrediccionConImg('t3.png')
label4,confidence4,topleft4,bottomright4,img4 = valoresPrediccionConImg('t4.png')
label5,confidence5,topleft5,bottomright5,img5 = valoresPrediccionConImg('t5.png')

#Mostramos los 4 en una linea
fig, axes = plt.subplots(1, 4, figsize=(10, 5))

axes[0].imshow(img)
axes[0].set_title(f'{label}\n Confianza: {confidence:.2f}')
axes[0].axis('off')

axes[1].imshow(img3)
axes[1].set_title(f'{label3}\n Confianza: {confidence3:.2f}')
axes[1].axis('off')

axes[2].imshow(img4)
axes[2].set_title(f'{label4}\n Confianza: {confidence4:.2f}')
axes[2].axis('off')

axes[3].imshow(img5)
axes[3].set_title(f'{label5}\n Confianza: {confidence5:.2f}')
axes[3].axis('off')

plt.tight_layout()
plt.show()
```

Podemos ver que con transformaciones básicas leves como una rotación de unos 15 grados la clasificación se mantiene estable y también en el caso de utilizar una imagen en blanco y negro, que en este caso mejora incluso un 1%, como efecto negativo con transformaciones muy radicales se pierde mucha confianza e incluso confunde el objeto con otro como es en el caso de la imagen 2.

Esto puede deberse a que el data augmentation realizado para entrenar el modelo empleaba transformaciones suaves por eso los cambios bruscos como en el caso de la segunda imagen de rotarla 180 grados provoca que no identifique el objeto

Procesar un video



Ahora que hemos visto una predicción en una imagen vamos a dar un paso más de dificultad haciendo lo mismo con un video, un video ya que no es más que una secuencia de imágenes.

Cabe decir que en este punto al tener una mayor carga de trabajo para el ordenador se va a notar al procesar el video

In [12]:

```
def valoresPrediccionResult(result):
    # cargamos la imagen, como al usar la función imread carga en BGR (Blue-Green-Re
    # Devuelve <class 'numpy.ndarray'> de la forma (x,y,3) Siendo el 3 los canales d
    # Obtenemos la predicción

    label = result['label']
    confidence = result['confidence']
    topleft = result['topleft']['x'],result['topleft']['y']
    bottomright = result['bottomright']['x'],result['bottomright']['y']

    # Devuelvo los 4 valores
    return label,confidence,topleft,bottomright


def getRectangleAndText(frame,label, topleft,bottomright,color):
    frame = cv2.rectangle(frame, topleft, bottomright, color, 7)
    frame = cv2.putText(frame, label, topleft, cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 25
    return frame

def randomNColors(n):
    colors = [tuple(255 * np.random.rand(3)) for i in range(n)]
    return colors
```

In [13]:

```
def valoresPrediccion(v):

    #Capturamos el video
    capture = cv2.VideoCapture(v)
    #Colores para las cajas
    colors = randomNColors(10)

    while (capture.isOpened()):
        # ret es True/False y frame es el Frame
        ret,frame = capture.read()
```

```
        # Hacemos la prediccion de ese frame
        result = tfnet.return_predict(frame)

        # Si se ha detectado el frame procedemos
        if ret != False:
            #con el zip mantengo colores si no se me ponen el mismo a todos
            for color, result in zip(colors, result):
                # LLamo al metodo anterior
                label,confidence,topleft,bottomright = valoresPrediccionResult(resul

                #Obtenemos el texto y el rectangulo del frame
                getRectangleAndText(frame,label, topleft,bottomright,color)

        #Vamos mostrando el frame
        cv2.imshow('frame', frame)
        #Para salir pulsar la tecla
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    #Destruimos las ventanas al terminar
    capture.release()
    cv2.destroyAllWindows()
```

In [14]:
```
valoresPrediccion('video/b.mp4')
```

In [15]:
```
display(Image(filename='img/bGifP.gif'))
```

```
<IPython.core.display.Image object>
```

Probemos ahora un video con múltiples objetos para comprobar como los va encuadrando

In [16]:
```
valoresPrediccion('video/cat.mp4')
```

In [17]:
```
display(Image(filename='img/catC.gif'))
```

```
<IPython.core.display.Image object>
```

Podemos utilizar las clasificaciones para filtrar

TFNet devuelve como hemos visto 4 valores, podemos utilizar el label y la confidence por ejemplo para filtrar para que solo muestre determinados objetos y filtrando este valor en el número de clases del modelo(COCO en este caso) para obtener un detector de objetos específicos

In [18]:
```
def valoresPrediccion(v):

    #Capturamos el video
    capture = cv2.VideoCapture(v)
    #Colores para las cajas
    colors = randomNColors(10)

    while (capture.isOpened()):
        # ret es True/False y frame es el Frame
        ret,frame = capture.read()
        # Hacemos la prediccion de ese frame
```

```
        result = tfnet.return_predict(frame)

        # Si se ha detectado el frame procedemos
        if ret != False:
            #con el zip mantengo colores si no se me ponen el mismo a todos
            for color, result in zip(colors, result):
                # LLamo al metodo anterior
                label,confidence,topleft,bottomright = valoresPrediccionResult(resul

                # Filtramos por pajaros con un nivel de confianza del 25%
                if(label == 'cat' and confidence > 0.25 ):

                    #Obtenemos el texto y el rectangulo del frame
                    getRectangleAndText(frame,label, topleft,bottomright,color)

        #Vamos mostrando el frame
        cv2.imshow('frame', frame)
        #Para salir pulsar la tecla
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    #Destruimos las ventanas al terminar
    capture.release()
    cv2.destroyAllWindows()
```

In [19]:
```
valoresPrediccion('video/cat.mp4')
```

In [20]:
```
display(Image(filename='img/catGF.gif'))
```

```
<IPython.core.display.Image object>
```

Procesar en tiempo real

Como hemos visto en la parte teórica YoloV2 es un modelo ligero pensado para transmisiones en vivo entre otras opciones, por lo que podiamos dejar de probar a realizar un par de pruebas usando la cámara de nuestro ordenador

In [21]:
```
def deteccionEnVivo():
    # definimos la camara que se usará, que en mi caso es la 0
    capture = cv2.VideoCapture(0)
    # Metodo de generar colores aleatorios
    colors = randomNColors(10)

    # Mientras este abierto
    while capture.isOpened():

        ret, frame = capture.read()
        if ret != False:
            # Hago la prediccion
            results = tfnet.return_predict(frame)
            for color, result in zip(colors, results):
                # Sacamos los valores del result
                label,confidence,topleft,bottomright = valoresPrediccionResult(resul
                #Obtenemos el texto y el rectangulo del frame
                getRectangleAndText(frame,label, topleft,bottomright,color)

            # Mostramos en una ventana emergente el frame
            cv2.imshow('frame', frame)
            # cerramos con la tecla
```

```
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

    capture.release()
    cv2.destroyAllWindows()
```

In [22]:
```
deteccionEnVivo()
```

In [23]:
```
display(Image(filename='img/vivoG.gif'))
```

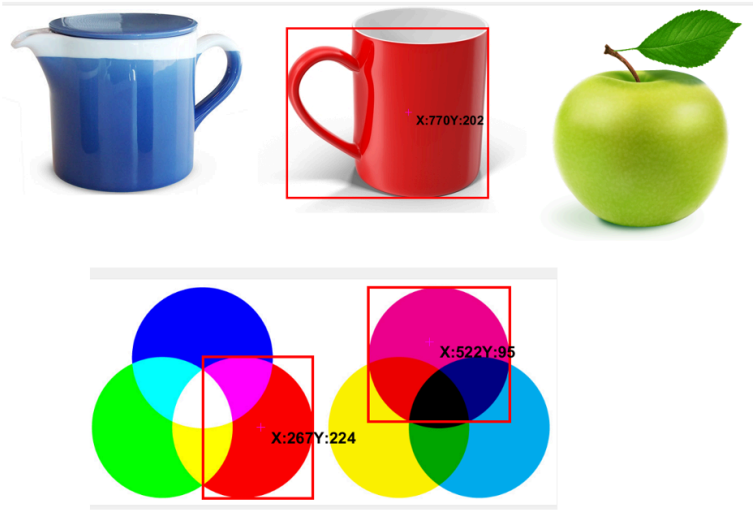
```
<IPython.core.display.Image object>
```

Detectar un rango de colores

Vamos a dar un paso más y añadir un paso más de dificultad filtrando en las transmisiones en vivo por color, para ello:

- Definiremos un rango de color que englobe el color que queremos capturar
- Seleccionaremos el area de interés(ROI) a través del area definida por la predicción
- Detectaremos el color del objeto y haremos la media para detectar el color promedio, intenté realizar con sklearn k medias para filtrar por color pero mi equipo no es lo suficientemente potente
- Comprobamos que el color del objeto cumple las restricciones
- si se cumple extraemos la máscara con la función inRange y la mostramos

Con todos estos pasos obtenemos usando el modelo de YoloV2 un método capaz de detectar objetos por categoría y tipo además del color



In [24]:
```
def detectaPorColorMask(color_range_low,color_range_high):

    # Tengo que convertirlo en array
    color_range_low = np.array(color_range_low)
    color_range_high = np.array(color_range_high)

    # Captura de video desde la cámara del portátil
```

```
    capture = cv2.VideoCapture(0)
    # Metodo para generar colores para las boxes
    colors = randomNColors(10)

    while capture.isOpened():
        # ret True y False - frame el frame actual
        ret, frame = capture.read()
        # Si detecta..
        if ret != False:
            #Prediccion
            results = tfnet.return_predict(frame)

            # iteramos por result y pasamos los colores
            for color, result in zip(colors, results):
                # sacamos los valores
                label,confidence,topleft,bottomright = valoresPrediccionResult(resul

                # Aqui podria especificarse un if para filtrar por etiqueta o valor
                # if (label == 'cup' and confidence >= 0.5):

                # Extraer el área del objeto detectado de interes (ROI)
                area_detectada = frame[topleft[1]:bottomright[1], topleft[0]:bottomr
                # Convertimos a HSV
                area_hsv = cv2.cvtColor(area_detectada, cv2.COLOR_BGR2HSV)
                # Media
                pixels = area_hsv.reshape((-1, 3))
                # Ordenamos los pixels y nos quedamos con la media
                mean_color = np.mean(pixels, axis=0)
                mean_color = tuple(int(c) for c in mean_color)
                # Comprobar si el color mediano está dentro del rango del color blan
                if (color_range_low <= mean_color).all() and (mean_color <= color_ra
                    # Dibujar rectángulo alrededor del objeto detectado
                    frame = getRectangleAndText(frame,label, topleft,bottomright,col
                    # Cargamos un texto adicional con el color
                    frame = cv2.putText(frame, f'Color: {mean_color}', (topleft[0],

                    # creamos la mascara para el color correspondiente
                    mask = cv2.inRange(area_hsv, color_range_low, color_range_high)

                    # Mostramos una ventana nueva cuando se detecte el valor de la m
                    cv2.imshow('mask', mask)

            # Mostramos el resultado en una ventana emergente
            cv2.imshow('frame', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        else:
            break

    capture.release()
    cv2.destroyAllWindows()
```

In [25]:

```
# Tonos de blanco
detectaPorColorMask([0, 0, 170], [180, 55, 255])
```

In [26]:

```
# Tonos de rojo
detectaPorColorMask([0, 50, 50], [10, 255, 255])
```

In [27]:

```
# Un rango de colores engloba todo
detectaPorColorMask([0, 0, 0],[255, 255, 255])
```

In [28]:
```
display(Image(filename='img/vivoG2.gif'))
```

```
<IPython.core.display.Image object>
```

Una breve introducción a Yolo9000

Primero cargamos el modelo como hemos hecho en YoloV2, debemos hacer esto ya que al disponer de 9000 clasificaciones diferentes los pesos , los nombres y la valoración de la predicción serán diferentes aunque usaremos la misma GPU para realizar las pruebas

In [29]:
```
configuration = {
    # Especificamos el modelo y su ruta en mi caso es YoloV2
    'model': 'cfg/yolo9000.cfg',
    # Cargamos los pesos
    'load': 'yolo9000-weights/yolo9000.weights',
    # Como de buena debe de ser la predicción para considerarla válida
    'threshold': 0.01,
    # Los nombres de las etiquetas
    'labels': 'cfg/9k.names',
    # 1 Para usar la GPU (en mi caso) y 0 para tirar de CPU
    'gpu': 1
}
tfnet9000 = TFNet(configuration)
```

```
Parsing ./cfg/yolo9000.cfg
Parsing cfg/yolo9000.cfg
Loading yolo9000-weights/yolo9000.weights ...
Successfully identified 195230020 bytes
Finished in 0.017030715942382812s
Model has name yolo9000, loading yolo9000 labels.

Building net ...
Source | Train? | Layer description                | Output size
-------+--------+----------------------------------+---------------
       |        | input                            | (?, 544, 544, 3)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 544, 544, 32)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 272, 272, 32)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 272, 272, 64)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 136, 136, 64)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 136, 136, 128)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 136, 136, 64)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 136, 136, 128)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 68, 68, 128)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 68, 68, 256)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 68, 68, 128)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 68, 68, 256)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 34, 34, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 34, 34, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 34, 34, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 34, 34, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 34, 34, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 34, 34, 512)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 17, 17, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 17, 17, 1024)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 17, 17, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 17, 17, 1024)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 17, 17, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 17, 17, 1024)
 Load  |  Yep!  | conv 1x1p0_1    linear           | (?, 17, 17, 28269)
-------+--------+----------------------------------+---------------
GPU mode with 1 usage
```

```
Finished in 7.587571620941162s
```

Como vimos en teoría el número de clasificaciones esta estructurado en forma de árbol dando predicciones ramificadas y en caso de no obtener una clasificación exacta la da del nodo anterior dando así una clasificación también correcta, vamos a ver un ejemplo con una imagen de un perro y después la compararemos con un resultado obtenido con YoloV2 con COCO en vez de Yolo9000

In [30]:
```
def showPredictions(img_path):
    # Cargamos la imagen en RGB
    img = cv2.cvtColor(cv2.imread(img_path, cv2.IMREAD_COLOR), cv2.COLOR_BGR2RGB)

    # Obtener las predicciones del modelo YOLO9000
    result = tfnet9000.return_predict(img)

    # Dibujar todas las cajas y etiquetas en la imagen
    for item in result:
        if item["label"] == "whole":
            continue
        topleft = (item['topleft']['x'], item['topleft']['y'])
        bottomright = (item['bottomright']['x'], item['bottomright']['y'])
        label = item['label']
        confidence = item['confidence']

        # dibujamos las cajas de las predicciones
        img = cv2.rectangle(img, topleft, bottomright, (0, 255, 0), 5)
        # Añadimos tambien texto a las cajas
        img = cv2.putText(img, f'{label} ({confidence:.2f})', topleft, cv2.FONT_HERS

    # mostramos el resultado
    plt.imshow(img)
    plt.axis('off')
    plt.show()
```

In [31]:
```
img = cv2.cvtColor(cv2.imread('img/sal.png', cv2.IMREAD_COLOR), cv2.COLOR_BGR2RGB)

result = tfnet9000.return_predict(img)

# Iteramos y mostramos los resultados
for item in result:
    if item["label"] != "whole":
        print(f"Clasificacion: {item['label']}, Confianza: {item['confidence']}")
```

```
Clasificacion: object, Confianza: 0.05900993570685387
Clasificacion: mammal, Confianza: 0.0832669660449028
Clasificacion: carnivore, Confianza: 0.037956539541482925
Clasificacion: canine, Confianza: 0.011308051645755768
Clasificacion: hunting dog, Confianza: 0.05337455868721008
```

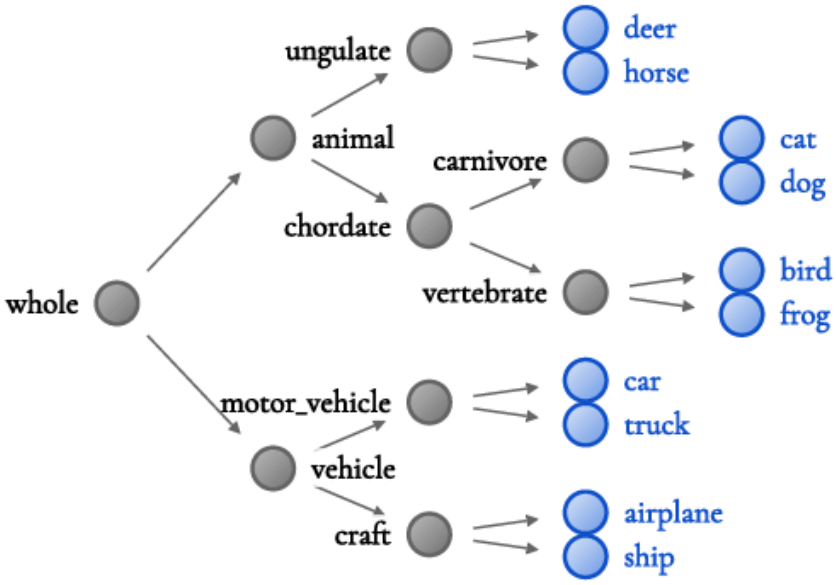
In [32]:
```
showPredictions('img/sal.png')
```

In [40]:
```
display(Image(filename='img/salgif.gif'))
```

```
<IPython.core.display.Image object>
```

In [34]:
```
def valoresPrediccion(v):

    #Capturamos el video
    capture = cv2.VideoCapture(v)
    #Colores para las cajas
    colors = randomNColors(10)

    while (capture.isOpened()):
        # ret es True/False y frame es el Frame
        ret,frame = capture.read()
        # Hacemos la prediccion de ese frame ahora con YOLO9000
        result = tfnet9000.return_predict(frame)

        # Si se ha detectado el frame procedemos
        if ret != False:
            #con el zip mantengo colores si no se me ponen el mismo a todos
            for color, result in zip(colors, result):
                # LLamo al metodo anterior
                label,confidence,topleft,bottomright = valoresPrediccionResult(resul

                #Obtenemos el texto y el rectangulo del frame
                if(label != 'whole'):
                    getRectangleAndText(frame,label, topleft,bottomright,color)

        #Vamos mostrando el frame
        cv2.imshow('frame', frame)
        #Para salir pulsar la tecla
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    #Destruimos las ventanas al terminar
    capture.release()
    cv2.destroyAllWindows()
```

In [36]:
```
valoresPrediccion('video/horse3.mp4')
```



Podemos ver el resultado en el siguiente GIF directamente, se puede apreciar que realiza múltiples clasificaciones para el mismo objeto como ya hemos ido comentando a lo largo de YOLO9000

En este caso podemos ver que para caballo realiza una clasificación de animal, chordate, ungulate y mamifero principalmente.

- Todas son correctas pues los ungulates por poner un ejemplo son mamiferos con pezuñas como caballos, renos...
- Chordate por ejemplo son animales con columna vertebal el cual tiene un caballo

Yolo9000 al tener tantas clasificaciones las predicciones no pueden ser muy altas por la gran variedad, pero abre la puerta a abrir entrenar modelos muy especificos con tantas clasificaciones como posee

In [39]:
```
display(Image(filename='img/horse3RSF.gif'))
```

Bilbiografia

https://github.com/philipperemy/yolo-9000

https://github.com/thtrieu/darkflow

https://es.mathworks.com/help/vision/ug/getting-started-with-yolo-v2.html

https://stackoverflow.com/questions/60674501/how-to-make-black-background-in-cv2-puttext-with-python-opencv

https://medium.com/@gastonace1/detecci%C3%B3n-de-objetos-por-colores-en-im%C3%A1genes-con-python-y-opencv-c8d9b6768ff

https://github.com/dwaithe/yolov2/blob/master/README.md

https://stackoverflow.com/questions/50390836/understanding-darknets-yolo-cfg-config-files

https://stackoverflow.com/questions/50005852/perform-multi-scale-training-yolov2

https://manalelaidouni.github.io/Understanding%20YOLO%20and%20YOLOv2.html

https://towardsdatascience.com/review-yolov2-yolo9000-you-only-look-once-object-detection-7883d2b02a65

https://leafmap.org/notebooks/76_image_comparison/

https://ieeexplore.ieee.org/document/8100173

https://github.com/Arup276/YOLOv2?tab=readme-ov-file

https://github.com/pjreddie/darknet

https://escholarship.org/content/qt0gg5d23s/qt0gg5d23s.pdf

https://tallerestampa.com/treballs/malalumne/jerarquia_yolo9000/

https://thinkinfi.com/color-detection-using-opencv-and-python/