Introduction to Yolo

YOLO is a popular object detection algorithm that can recognize and localize multiple objects within an image or video in real-time. The YOLO network uses a single convolutional neural network (CNN) architecture to simultaneously perform object detection and classification.

The YOLO network divides an input image into a grid of cells and predicts bounding boxes, objectness scores, and class probabilities for each cell. Each bounding box is represented by five values: the x and y coordinates of the box's center, the box's width and height, and a confidence score that indicates the probability that the box contains an object. The objectness score indicates the probability that an object is present in the box, and the class probabilities represent the probabilities of the box containing objects of different classes.

The YOLO network uses a variant of the Darknet architecture, which consists of 53 convolutional layers followed by several fully connected layers. The network architecture is designed to optimize the trade-off between accuracy and speed, allowing the algorithm to achieve real-time performance even on low-powered devices.

Training the YOLO network involves using a large dataset of annotated images to optimize the network's parameters through backpropagation. During training, the network learns to predict bounding boxes and class probabilities that accurately match the ground truth annotations.

Overall, YOLO is a highly effective and efficient object detection algorithm that has been widely used in a variety of applications, including autonomous vehicles, surveillance systems, and image and video analysis.

Yolo 7

YOLOv7 provides a greatly improved real-time object detection accuracy without increasing the inference costs. When compared to other known object detectors, YOLOv7 can effectively reduce about 40% parameters and 50% computation of state-of-the-art real-time object detections, and achieve faster inference speed and higher detection accuracy. In general, YOLOv7 provides a faster and stronger network architecture that provides a more effective feature integration method, more accurate object detection performance, a more robust loss function, and an increased label assignment and model training efficiency. As a result, YOLOv7 requires several times cheaper computing hardware than other deep learning models. It can be trained much faster on small datasets without any pre-trained weights.

Clone Yolo v7 repo

The repository being cloned is a fork of the Yolov7 repository with customized settings. The following settings have been customized for the purpose of this project:

- `cfg/training/yolov7.yaml` update the classes number to 52
- `data/coco.yaml` : changed the path of the datasets and updated the number and value of the classes
- `data/data.yaml` : it's a copy of the coco.yaml file; such copy was created because using the original file would trigger some tasks specific for the coco dataset
- `requirements.txt` : replace the default torch version with the CUDA enabled one

```
In [2]:  !git clone https://github.com/Filocava99/Yolov7-Poker-Hands-classification.git yolo
```

```
fatal: destination path 'yolov7' already exists and is not an empty directory.
```

Libraries import

Before training the model all the required modules have to be installed

```
In [ ]:  !pip install -r .//yolov7//requirements.txt
```

To install pytorch with CUDA support run the following cell:

```
In [ ]:  !pip install torch==1.13.1+cu116 torchvision==0.14.1+cu116 torchaudio==0.13.1 --ext
```

```python
In [4]:  import requests, zipfile, io, shutil, os, torch, pycocotools, yaml
         import pandas as pd
         import ipywidgets as widgets
         import numpy as np
         import cv2
```

Downloading the compiled Yolo v7 model and its less complex version (not used in the final/production version)

```python
In [5]:  yolo_v7 = "https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7_train
         yolo_v7_tiny = "https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7-
         r = requests.get(yolo_v7)
         with open('./yolov7/yolov7.pt', "wb") as f:
             f.write(r.content)
         r = requests.get(yolo_v7_tiny)
         with open('./yolov7/yolov7_tiny.pt', "wb") as f:
             f.write(r.content)
```

Dataset import

The following cell will download the dataset from a Google drive url. In case the url has expired you can generate a new one from the commented Kaggle link (second line of code). After the dataset has been downloaded, all the classes described in the `data.yaml` file will be parsed for later use and then printed.

```
In [10]:  obj_detection_dataset_url = "https://storage.googleapis.com/kaggle-data-sets/104367
          #"https://drive.google.com/uc?export=download&id=1GOJ9REocmncYz5Fc_bXgF3vb313EJrsh"
          #"https://www.kaggle.com/datasets/andy8744/playing-cards-object-detection-dataset/d

          if(os.path.exists("./yolov7/dataset") == False):
              r = requests.get(obj_detection_dataset_url)
              z = zipfile.ZipFile(io.BytesIO(r.content))
              z.extractall("./yolov7/dataset")

          from yaml.loader import SafeLoader
          classes = []
          # Open the file and load the file
          with open('./yolov7/dataset/data.yaml') as f:
              classes = yaml.load(f, Loader=SafeLoader)["names"]
          print("Total classes: " + str(len(classes)))
          print("Classes: " + str(classes))
```

```
Total classes: 52
Classes: ['10c', '10d', '10h', '10s', '2c', '2d', '2h', '2s', '3c', '3d', '3h', '3
s', '4c', '4d', '4h', '4s', '5c', '5d', '5h', '5s', '6c', '6d', '6h', '6s', '7c', '7
d', '7h', '7s', '8c', '8d', '8h', '8s', '9c', '9d', '9h', '9s', 'Ac', 'Ad', 'Ah', 'A
s', 'Jc', 'Jd', 'Jh', 'Js', 'Kc', 'Kd', 'Kh', 'Ks', 'Qc', 'Qd', 'Qh', 'Qs']
```

Important

> The following steps can be skipped if you want to use the already trained
> model, otherwise be aware that the training phase can require up to 4 hours
> to complete (tested on an RTX 4080).

Remove roboflow junk from file names

The dataset creator used roboflow to generate both the labels and the images, so we have
to strip such files from unneeded information, especially in their files name.

```
In [19]:  import os

          # remove roboflow extra junk

          count = 0
          for i in sorted(os.listdir('./yolov7/dataset/train/labels')):
              if count >=3:
                  count = 0
              count += 1
              if i[0] == '.':
                  continue
              j = i.split('_')
              dict1 = {1:'a', 2:'b', 3:'c'}
              source = './yolov7/dataset/train/labels/'+i
              dest = './yolov7/dataset/train/labels/'+j[0]+dict1[count]+'.txt'
              os.rename(source, dest)
```

```python
count = 0
for i in sorted(os.listdir('./yolov7/dataset/train/images')):
    if count >=3:
        count = 0
    count += 1
    if i[0] == '.':
        continue
    j = i.split('_')
    dict1 = {1:'a', 2:'b', 3:'c'}
    source = './yolov7/dataset/train/images/'+i
    dest = './yolov7/dataset/train/images/'+j[0]+dict1[count]+'.jpg'
    os.rename(source, dest)

for i in sorted(os.listdir('./yolov7/dataset/valid/labels')):
    if i[0] == '.':
        continue
    j = i.split('_')
    source = './yolov7/dataset/valid/labels/'+i
    dest = './yolov7/dataset/valid/labels/'+j[0]+'.txt'
    os.rename(source, dest)

for i in sorted(os.listdir('./yolov7/dataset/valid/images')):
    if i[0] == '.':
        continue
    j = i.split('_')
    source = './yolov7/dataset/valid/images/'+i
    dest = './yolov7/dataset/valid/images/'+j[0]+'.jpg'
    os.rename(source, dest)
for i in sorted(os.listdir('./yolov7/dataset/test/labels')):
    if i[0] == '.':
        continue
    j = i.split('_')
    source = './yolov7/dataset/test/labels/'+i
    dest = './yolov7/dataset/test/labels/'+j[0]+'.txt'
    os.rename(source, dest)

for i in sorted(os.listdir('./yolov7/dataset/test/images')):
    if i[0] == '.':
        continue
    j = i.split('_')
    source = './yolov7/dataset/test/images/'+i
    dest = './yolov7/dataset/test/images/'+j[0]+'.jpg'
    os.rename(source, dest)
```

Train the dataset

To train the dataset we use the following parameters:

- workers: 8, adjust to the number of CPU cores
- device: 0, where 0 is the id of the GPU
- batch size: 32
- epochs: 100
- img: 416 416, the size of the images of the dataset

- hyp: data/hyp.scratch.custom.yaml, the file containing the hyperparameters to use
- name: yolov7-custom, the name to use to save the trained model
- weights: yolov7.pt, the weights to start from

```
In [ ]: !python .//yolov7//train.py --workers 8 --device 0 --batch-size 32 --epochs 100 --i
```

After the training we can run the model to detect the cards on different sources. In the first case we are providing a video while in the second case we are using the webcam feed stream.

```
In [ ]: !yt-dlp --format mp4 "https://youtu.be/koxA7TqdNWk"
        !python .//yolov7//detect.py --weights .//yolov7//runs//train//yolov7-custom//weigh
```

```
In [ ]: !python .//yolov7//detect.py --weights .//yolov7//runs//train//yolov7-custom//weigh
```

Test evaluation

```
In [25]: # Load the image
         image_path = './yolov7/runs/train/yolov7-custom/results.png'
         with open(image_path, 'rb') as f:
             image = f.read()

         # Create the image widget
         image_widget = widgets.Image(
             value=image,
             format='png',
             width=1920
         )
         display(image_widget)
```

Image(value=b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\t`\x00\x00\x04\xb0\x08\x06\x00\x00\x00\x7f\xa4q\x81\...

I grafici mostrati nella cella precedente sono stati generati direttamente dall'implementazione di Yolo 7 che è stata utilizzata. Sono di particolare interesse i grafici di precision, recall e mAP, sia con IoU threshold 0.5 che 0.95. The graphs shown in the previous cell were generated directly from the implementation of Yolo 7 that was used. Of particular interest are the precision, recall, and mAP graphs, both with an IoU threshold of 0.5 and 0.95.

The model converged around the 30th epoch, but we continued training up to 100 epochs to verify any potential improvements. The mAP improved slightly, but there were no issues of overfitting, so overall the subsequent epochs were not considered useless.

Score evaluation algorithm

This algorithm will return a tuple of two elements, the first element is an integer between 1 and 10 representing the point value of the hand and the second element is the highest card in case of tie-breaker. The point values are:

- 10 for Straight Flush
- 8 for Four of a Kind
- 7 for Full House
- 6 for Flush
- 5 for Straight
- 4 for Three of a Kind
- 3 for Two Pairs
- 2 for One Pair
- 1 for High Card

The expected input for this algorithm is a list of five strings, where each string represents a poker card in the format of "value suit". The value can be one of the following: '2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A'. The suit can be one of the following: 'C' for Clubs, 'D' for Diamonds, 'H' for Hearts, 'S' for Spades.

```
cards = ["2 H", "3 H", "4 H", "5 H", "6 H"]
poker_hand(cards)
```

In [26]:
```python
import collections

points = {
    10: "Straight flush",
    8: "Four of a kind",
    7: "Full house",
    6: "Flush",
    5: "Straight",
    4: "Three of a kind",
    3: "Two pairs",
    2: "One pair",
    1: "High card"
}

def poker_hand(cards):
    # Create a dictionary to store the count of each card value
    card_count = collections.defaultdict(int)
    for card in cards:
        value = card[0]
        if value == 'T':
            value = '10'
        elif value == 'J':
            value = '11'
        elif value == 'Q':
            value = '12'
        elif value == 'K':
            value = '13'
        elif value == 'A':
            value = '14'
        card_count[value] += 1
    # Check for a flush (all cards are the same suit)
    flush = all(card[2] == cards[0][2] for card in cards)
    # Check for a straight (cards are in sequential order)
    straight = (max([int(x) for x in card_count.keys()]) - min([int(x) for x in car
```

```python
        # Check for a straight flush (flush and straight)
        straight_flush = flush and straight
        # Check for four of a kind
        four_of_a_kind = any(count == 4 for count in card_count.values())
        # Check for three of a kind
        three_of_a_kind = any(count == 3 for count in card_count.values())
        # Check for a full house (three of a kind and one pair)
        full_house = three_of_a_kind and any(count == 2 for count in card_count.values(
        # Check for two pairs
        two_pairs = len([count for count in card_count.values() if count == 2]) == 2
        # Check for one pair
        one_pair = len([count for count in card_count.values() if count == 2]) == 1
        if straight_flush:
            return (points[10], max(card_count.keys()))
        elif four_of_a_kind:
            return (points[8], max(card_count, key=lambda x: card_count[x]))
        elif full_house:
            return (points[7], max(card_count, key=lambda x: card_count[x]))
        elif flush:
            return (points[6], max(card_count.keys()))
        elif straight:
            return (points[5], max(card_count.keys()))
        elif three_of_a_kind:
            return (points[4], max(card_count, key=lambda x: card_count[x]))
        elif two_pairs:
            return (points[3], max(card_count, key=lambda x: card_count[x]))
        elif one_pair:
            return (points[2], max(card_count, key=lambda x: card_count[x]))
        else:
            return (points[1], max(card_count.keys()))
```

You can select a photo of a poker hand to test out the model:

```python
In [27]:  uploader = widgets.FileUpload(multiple=False)
          display(widgets.HBox([uploader]))
```

HBox(children=(FileUpload(value=(), description='Upload'),))

```python
In [28]:  fileName = list(uploader.value[0].items())[0][1]
          content = list(uploader.value[0].items())[3][1]
          widgets.Image(value=content)
```

Out[28]: Image(value=b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x00H\x00H\x00\x00\xff\x
          e2\x02(ICC_PROFILE\x00\x01\x…

```python
In [29]:  decoded = cv2.imdecode(np.frombuffer(content, np.uint8), -1)
          width = decoded.shape[1]
          height = decoded.shape[0]
          max_size = max(width, height)
          print("Width: " + str(width))
          print("Height: " + str(height))
          print("If you want to use the webcam feed instead of the uploaded image please chec
          checkbox = widgets.Checkbox(
              value=False,
              description='Use webcam',
              disabled=False
```

```
    )
display(checkbox)
```

```
Width: 720
Height: 1280
If you want to use the webcam feed instead of the uploaded image please check the fo
llowing checkbox:
Checkbox(value=False, description='Use webcam')
```

In [30]:
```python
if(os.path.exists("./runs/detect/custom")):
    shutil.rmtree("./runs/detect/custom")
os.makedirs("./runs/detect/custom")
with open("./runs/detect/custom/test.jpg", "wb") as f:
    f.write(content)
if(checkbox.value):
    !python .//yolov7//detect.py --weights .//yolov7//runs//train//yolov7-custom//w
else:
    !python .//yolov7//detect.py --weights .//yolov7//runs//train//yolov7-custom//w
```

```
Namespace(weights=['.//yolov7//runs//train//yolov7-custom//weights//best.pt'], sourc
e='.//runs//detect//custom//test.jpg', img_size=1280, conf_thres=0.5, iou_thres=0.4
5, device='', view_img=True, save_txt=True, save_conf=False, nosave=False, classes=N
one, agnostic_nms=False, augment=False, update=False, project='runs/detect', name='c
ustom', exist_ok=True, no_trace=False)
Fusing layers...
 Convert model to Traced-model...
 traced_script_module saved!
 model is traced!

1 5d, 1 6d, 1 7h, 1 8d, 1 9c, Done. (27.5ms) Inference, (3.5ms) NMS
 The image with the result is saved in: runs\detect\custom\test.jpg
Done. (0.225s)
```
```
YOLOR  2023-5-11 torch 1.13.1+cu116 CUDA:0 (NVIDIA GeForce RTX 4080, 16375.375MB)

C:\Users\filip\anaconda3\envs\visione\lib\site-packages\torch\functional.py:504: Use
rWarning: torch.meshgrid: in an upcoming release, it will be required to pass the in
dexing argument. (Triggered internally at ..\aten\src\ATen\native\TensorShape.cpp:31
91.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model Summary: 200 layers, 6144193 parameters, 0 gradients, 13.5 GFLOPS
```

In [31]:
```python
df = pd.read_csv("./runs/detect/custom/labels/test.txt", delim_whitespace=True, hea
objs = []
for index, row in df.iterrows():
    name = str(classes[int(row[0])])
    name = name[0] + " " + str.upper(name[1])
    if((name in objs) == False):
        objs.append(name)
print(objs)
```

```
['8 D', '7 H', '5 D', '9 C', '6 D']
```

In [32]:
```python
hand = poker_hand(objs)
print("Point: " + hand[0])
print("Highest card value: " + str(hand[1]))
```

```
Point: Straight
Highest card value: 9
```

Final thoughs

The Yolo v7 model outperfomed the SSD model reaching a mAP of over 95%, while SSD only topped 5%. There are still a few issues, probably related to the dataset used; in particular, the model struggles to detect the values of the cards when the photo has been taken to close to them (some background has to be visible for the model to detect cards). Another issue, which is instead caused by the scoring algorithm, is that the same card may be counted twice when all of its corners are uncovered; as a design implementation I decided not to handle it, to better showcase how the model predictions work.