0.1 GCD and Bezout's identity

- *Algorithm:* name (algo. 1)
- *Input:* two integers a and b
- *Complexity:* complexity of the algorithm, e.g. $\mathcal{O}(\log a)$
- *Data structure compatibility:* N/A
- *Common applications:* Number theory

Problem. GCD and Bezout's identity

Given two integers a and b, calculate the GCD of a and b as d. Bezout's identity is that there exist two integers x and y such that $ax + by = d$.

Description

1. Bezout's identity

Bezout's identity is a fundamental theorem in the field of number theory, which is described as follow: given two integers a and b, assume the greatest common divider(GCD) of them is d, there exist two integers x and y, such that $ax + by = d$.

I will give my own proof here. Since the GCD of a and b is d, we can assume $a = d \times p$ and $b = d \times q$, where $(p, q) = 1$. Then we need to find x and y such that $px + qy = 1$. This is equivalent to $p|qy - 1$, which is to find y such that $qy \equiv 1 \mod p$. Since $(y, p) = 1$, we can always find such q. Qed. In the last step we can also let $q = 0, 1, \cdots, p - 1$, then for these q, $qy \equiv r \mod p$. r is different as q becomes different. So there must be a q such that $qy \equiv 1 \mod p$.

Notice that Bezout's identity can be generalized into more than two integers. Given n integers $a_1, a_2, \cdots, a_n$, assume the GCD of them is d, then there exists integers $x_1, x_2, \cdots, x_n$, such that $d = a_1x_1 + a_2x_2 + \cdots + a_nx_n$. The proof using induction is not as straightforward as the proof for two-integer case. For finite field, we have similar Bezout's identity. Example of calculating GCD on finite field can be found in the slides provided on Friday's VE477 lecture[2].

This is my own naive proof using induction. For input of size two, we have already proved above. Now assume Bezout's identity is true for k inputs, and we need to prove that it is also true for $k + 1$ inputs. Assume the $k + 1$ inputs are $a_1, a_2, \cdots, a_k, a_{k+1}$, GCD of first k inputs as d, GCD of first $k + 1$ inputs as e, it is obvious that we can write $d = e \times f$ and $(a_{k+1}, f) = 1$. Using the induction hypothesis, there exists $x_1, x_2, \cdots, x_k$, such that $a_1x_1 + a_2x_2 + \cdots + a_kx_k = d$. We only need to find x_{k+1} such that $d + a_{k+1}x_{k+1} = e$. This is equivalent to $f + \frac{a_{k+1}}{e}x_{k+1} = 1$, since $(a_{k+1}, f) = 1$, because of the Bezout's identity with 2 inputs, there exist such x_{k+1}, Qed. For proof in finite field, maybe in the final exam, who knows.

2. GCD

When we need to take advantage of the Bezout's identity, we need to calculate the GCD of two different integers first. Here GCD problem is that given two integers a and b, calculate the largest positive integer d such that d divide both a and b. Notice that we can define the GCD of more than two integers in a similar way. The

commonly used algorithm for calculating GCD is Euclidean algorithm. Other common applications of GCD are extended Euclidean algorithm, linear Diophantine equations, and Chinese remainder theorem.

Euclidean algorithm is based on subtracting the smaller input from the larger one until one of the input becomes zero. We can speed up the process by using division instead of subtraction in some cases. The number of divisions required for Euclidean algorithm is $\mathcal{O}(\log(min(a, b)))$, where a, b are the input integers[1]. Thus we have the time complexity of $\mathcal{O}(\log(min(a, b)))$.

Algorithm 1: Euclidean algorithm

Input : two integers a and b
Output: the GCD of a and b

```
while b ≠ 0 do
    if a > b then
        a ≡ tmp mod b
        a = tmp
    end if
    else
        b ≡ tmp mod b
        b = tmp
    end if
end while
return a
```

References.

[1] H Grossman. "Discussions: On the number of divisions in finding a GCD". In: *The American Mathematical Monthly* 31.9 (1924), pp. 443–443 (cit. on p. 2).

[2] Manuel. *VE477 – Introduction to Algorithms (lecture slides)*. 2019 (cit. on p. 1).

0.2 Vertex independent set

- *Algorithm:* greedy algorithm (algo. 3)
- *Input:* An undirected graph $G(V, E)$
- *Complexity:* $\mathcal{O}(n^3)$
- *Data structure compatibility:* Graph can be represented by either adjacency matrix(dense graph) or adjacency list(sparse graph)
- *Common applications:* Theory computer science

Problem. Vertex independent set

Given an undirected graph $G(V, E)$, and a subset V' of V is vertex independent set if and only if for any two different nodes u, v in V', there is no edge between them.

Description

We have studied the clique problem in the homework. Vertex independent set problem, however, is somewhat the opposite of clique. We have an undirected graph $G(V, E)$, a subset V' of V is called vertex independent set if and only if for any two different nodes u, v in V', there is no edge between them.

Moreover, we are interested in maximal independent set(MIS) as well. Here MIS is defined as the vertex independent set with largest cardinality in the graph. We are interested in finding algorithms to find such an MIS.

MIS problem has many theoretical applications and can be used to prove the time complexity of many other theoretical problems in the field of theory computer science. For example, we can use the linear programming(LP) knowledge we learned from lab 8 and transform the MIS problem into its dual which is minimum set cover(MSC) problem. Usually we can reduce some hard problem to the MIS and prove its complexity.

We first try with brute force. Assume there are n nodes in the graph. We need to list all the possibilities of choosing nodes in the graph. You can either choose a node or not. There should be $\mathcal{O}(2^n)$ choices. After choosing, we need to test all the edges between them, which takes $\mathcal{O}(n^2)$ time because there are $n^2/2$ pairs to check. The pseudo code is in the first algorithm below. In 2017, it has been proved that there exists an exact algorithm for MIS with time complexity of $\mathcal{O}(1.1996^n)$ using polynomial space, where n is the size of nodes in the graph[2]. Although this method is much better than brute force, it is still unachievable to get an exact answer for the problem. To tackle the MIS problem, we need to go with approximation algorithms.

One good choice is to go with greedy. We choose the node v with smallest degree and add to a set. After that, we delete all the neighbors of v from the graph G. We repeat this until G becomes empty. The pseudo code is in the second algorithm below. To find the node with maximum degree, we need to check all the edges, which takes $\mathcal{O}(n^2)$. Then we need to do this for $\mathcal{O}(n)$ nodes in worst case. Thus the overall time complexity for greedy is $\mathcal{O}(n^3)$. The polynomial time complexity is quite satisfactory for us, but the sad story is that we have no guarantee on the quality of the MIS we get using greedy algorithm. It is proved by Bazgan in 2004 that the MIS problem is Poly-APX-complete and thus no approximation algorithm can achieve a constant factor in polynomial time[1].

References.

[1] Cristina Bazgan, Bruno Escoffier, and Vangelis Th Paschos. "Completeness in standard and differential approximation classes: Poly-(D) APX-and (D) PTAS-completeness". In: *Theoretical Computer Science* 339.2-3 (2005), pp. 272–292 (cit. on p. 3).

[2] Mingyu Xiao and Hiroshi Nagamochi. "Exact algorithms for maximum independent set". In: *Information and Computation* 255 (2017), pp. 126–146 (cit. on p. 3).

0.3 Matrix multiplication

- *Algorithm:* Naive (algo. 4 & alg:5), Strassen (algo. algo. 6)
- *Input:* Matrices
- *Complexity:* $\mathcal{O}(n^\omega)$, with $2 < \omega$
- *Data structure compatibility:* Array (2D matrix)

Algorithm 2: brute force

Input : An undirected graph $G(V, E)$
Output: MIS of $G(V, E)$

```
n = |V|
MIS = 0
size = 0
for i = 0; i < 2^n; i++ do
    correct = True
    represent i = a_1, a_2, ··· , a_n in base 2
    for j = 1; j < n + 1; j++ do
        for k = 1; k < n + 1; k++ do
            if a_j and a_k and (j, k) ∈ E then
                correct = False
            end if
        end for
    end for
    if correct and sum(a_1, a_2, ··· , a_n) > size then
        size = sum(a_1, a_2, ··· , a_n)
        MIS = i
    end if
end for
return S
```

Algorithm 3: greedy algorithm

Input : An undirected graph $G(V, E)$
Output: MIS of $G(V, E)$

```
S = ∅ while G is not empty do
    v be node of minimum degree in G
    S = S ∪ v
    remove v and its neighbours from G
end while
return S
```

- *Common applications:* All kinds of calculation missions in Mathematics, Physics, Engineering, etc.

Problem. Matrix multiplication

The multiplication between matrices is a common calculation mission. A naive way to calculate it according to the definition of matrix multiplication is of high complexity. Therefore several algorithms of better complexity performances are proposed, which gradually cut the time complexity to a polynomial of lower power. The basic ideas of those algorithms will be discussed here.

Description

Firstly recall the definition of matrix multiplication. Given 2 matrices A, B of size $m \times p$, $p \times n$, their product M is a $m \times n$ matrix with elements

$$M_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj}$$

for any $1 \leq i \leq m$ and $1 \leq j \leq n$. Therefore the most instinctive method is calculate the elements in M one by one.

It is easy to see that the Algo. 4 cost $\Theta(mpn)$. For purpose of generality, we may assume the both matrices are square matrices of size $n \times n$, and the time complexity is then $\mathcal{O}(n^3)$.

Algorithm 4: Naive (Iteration)

Input : matrices A of $m \times p$ and B of size $p \times n$
Output: matrix M the product of A and B

```
for i for 1 to m do
    for j for 1 to n do
        M_ij ← 0
        for k for 1 to p do
            M_ij ← M_ij + A_ik B_kj
        end for
    end for
end for
return M
```

There is an *divide-and-conquer alternative* for it, based on which further improvements are possible. The basic idea here is *block partition*, to keep on splitting the original large matrices into halves so as to consecutively divide the problem into multiplications of 2×2 square matrices. Suppose $C = AB$, where

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}, A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}.$$

and we may calculate C in the way of

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

It means that we only need to calculate 4 smaller-size multiplications and n^2 additions. The complexity is then given by recurrence

$$\begin{cases} T(1) = \Theta(1) \\ T(n) = 8T(n/2) + \Theta(n^2) \end{cases}$$

We can see that the time complexity is $\mathcal{O}(n^3)$ according to Master Theorem, which the same as the iteration method. The pseudocode is given in Algo. 5

Algorithm 5: Name

Input : $n \times n$ square matrices A, B
Output: M the product of A, B

```
Function Multi(A, B):
    if A, B is 2 × 2 then return AB        /* 8 scalar multiplications */ ;
    divide A = [A11 A12; A21 A22], B = [B11 B12; B21 B22]
    M1 ← Multi(A11, B11)
    M2 ← Multi(A12, B21)
    M3 ← Multi(A11, B12)
    M4 ← Multi(A12, B22)
    M5 ← Multi(A21, B11)
    M6 ← Multi(A22, B21)
    M7 ← Multi(A21, B12)
    M8 ← Multi(A22, B22)
    calculate C = [M1 + M2  M3 + M4; M5 + M6  M7 + M8]
end
return C
```

Strassen Algorithm is named after Volker Strassen, proposed in 1969, it is one of the important tries to decrease the matrix multiplication complexity [3]. Although the improvement of Strassen Algorithm is limited, it inspires later inventions of better algorithms. The basic idea of Strassen Algorithm is similar to that of divide-and-conquer scheme, but cut the number of sub-problems in each layer of recursion. With the same background settings as in Algo. 2, the calculation is altered as in Algo. 3.

Algorithm 6: Strassen

Input : $n \times n$ square matrices A, B
Output: M the product of A, B

Function `Multi(`A, B`)`**:**
  if A, B *is* 2×2 **then return** AB `/* 8 scalar multiplications */` ;
  divide $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$
  $M_1 \leftarrow Multi(A_{11} + A_{22}, B_{11} + B_{22})$
  $M_2 \leftarrow Multi(A_{21} + A_{22}, B_{11})$
  $M_3 \leftarrow Multi(A_{11}, B_{12} - B_{22})$
  $M_4 \leftarrow Multi(A_{22}, B_{21} + B_{11})$
  $M_5 \leftarrow Multi(A_{11} + A_{12}, B_{22})$
  $M_6 \leftarrow Multi(A_{21} - A_{11}, B_{11} + B_{12})$
  $M_7 \leftarrow Multi(A_{12} - A_{22}, B_{21} + B_{22})$
  calculate $C = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$
end
return C

Now the recurrence is

$$\begin{cases} T(1) = \Theta(1) \\ T(n) = 7T(n/2) + \Theta(n^2) \end{cases}$$

and hence the complexity is $\mathcal{O}(n^{\log_2 7}) = \mathcal{O}(n^{2.807})$. Although the calculation process is more complicated compared to naive algorithms, Strassen Algorithm has better performances in cases where $n > 100$ [1].

Since any algorithm managing to deal with matrix multiplication has to traversal all the n^2 entries in two matrices, the time complexity of matrix multiplication algorithm has a lower bound $\mathcal{O}(n^2)$. Setting the $\mathcal{O}(n^3)$ naive algorithm has baseline, we can express the time complexity of all matrix multiplication algorithms as $\mathcal{O}^\omega$, where ω is called the exponent of matrix multiplication, which is a number between 2 and 3. The lowest known k is achieved by François Le Gall with Coppersmith–Winograd algorithm, which is 2.3728639 [2].

References.

[1] Michael A Bender, Dongdong Ge, Simai He, Haodong Hu, Ron Y Pinter, Steven Skiena, and Firas Swidan. "Improved bounds on sorting by length-weighted reversals". In: *Journal of Computer and System Sciences* 74.5 (2008), pp. 744–774 (cit. on p. 6).

[2] François Le Gall. "Powers of tensors and fast matrix multiplication". In: *Proceedings of the 39th international symposium on symbolic and algebraic computation*. 2014, pp. 296–303 (cit. on p. 6).

[3] Volker Strassen. "Gaussian elimination is not optimal". In: *Numerische Mathematik* 13 (Aug. 1969), pp. 354–356 (cit. on p. 6).

0.4 PageRank

- *Algorithm:* PageRank (algo. 7)

- *Input:* A network/graph G with N nodes
- *Complexity:* $\mathcal{O}(k * N)$, where k is #iteration
- *Data structure compatibility:* Graph
- *Common applications:* Web pages ranking.

Problem. PageRank

In a network, we measure the importance of a node with *rank*. The *PageRank* algorithm is a Monte-Carlo method to estimate the rank of all nodes in a network. It sets a uniform prior distribution for ranks and recursively updates it with a network transition matrix obtained from node connections.

Description

A network is composed of nodes, and each of them has its own portion of significance. In many cases, we would prefer to visit the more important nodes. For example, among millions of entries in the result of search engines, those pages with higher weights will be shown in the first few pages, which are considered of higher quality. We measure the importance of a node with *rank*. A page with higher rank has more importance in the network.

PageRank is developed by Google company to rank web pages in their search engine results. Is is named after *Larry Page*, one of the co-founder of the company [3]. PageRank estimates the importance of an web page by observing the number and quality of links to it. The idea is based on two assumptions:

- A page if of more importance if it has more links pointing in.
- A link from an important page add more to a page's own importance.

Based on the idea, we may have the following way to calculate the rank of a page. Denote r_i the rank of node i in a network, and we have

$$r_i = \sum_{j \in \mathcal{N}(i)} \frac{r_j}{d_j}$$

where $\mathcal{N}(i)$ means the nodes linking into i and d_j is the out degree of node j. The equations for all nodes form a linear equation system, and we can express it in the way of

$$\mathbf{r} = M \cdot \mathbf{r}$$

where matrix M is defined as

$$M_{ij} = \begin{cases} 1/d_j, \text{ if } j \to i \\ 0, \text{ otherwise} \end{cases}$$

Therefore, to find the page ranks is to find the eigenvector of the matrix M. However, to efficiently solve for $\mathbf{r}$, we would adopt power iteration. As a Monte-Carlo Markov Chain (MCMC), the ranks is the stationary distribution of the matrix M, and the answer will approach it no matter what initial value we assign to $\mathbf{r}$. We will follow the scheme below:

1. Initialize $\mathbf{r}^(0) = [r_1, \cdots, r_N]$ with $r_i = 1/N$ for all $i \in [1, N]$.
2. Update $\mathbf{r}^{(t+1)} = M \cdot \mathbf{r}^{(t)}$
3. Repeat step 2 until $|r^{(t+1)} - r^{(t)}| < \eta$ a small positive number.

There are two main problem with this simple form. Firstly, if the out-degree of node j is 0, i.e. it is linked to no other pages, then all $M_{.j} = 0$, causing the chain to enter a dead end. To deal with it, simply assign a uniform distribution to the j-th row.

Algorithm 7: PageRank

Input : Network with N nodes, small positive number η
Output: ranks $\mathbf{r}$ for all nodes

```
for i ← 1 to N do
    for j ← 1 to N do
        if j points to i then M_ij = 1/degree_{j,out};
        else M_ij = 0;
    end for
end for
for i ← 1 to N do
    r_i^0 ← 1/N
end for
while |r^(t+1) − r^(t)| < η do
    r^(t+1) = M · r^(t)
end while
return r
```

Another problem is when a group of one or more pages have no links pointing out of the group, most of the ranks would be adsorbed by that group. To avoid it, we introduce random jump. Instead of always following the current ranks, the random walk jump to a node according to a uniform distribution with a small probability $1-\beta$. This help jump out of such rank "drains". The update rule now written as [1]

$$r_i = \sum_{j \to i} \beta \frac{r_j}{d_j} + (1-\beta)\frac{1}{N}$$

and this leads to the Google Matrix that Google adopts

$$A = \beta M + (1-\beta) + \frac{1}{N} I_{1/N}.$$

The complexity of PageRank algorithm is $\mathcal{O}(kN)$, where k is the number of iterations. We have known that a matrix-vector multiplication costs $(O)(n^2)$, then the total complexity is originally larger. However, as a sparse network, the matrix M have few non-zero elements, and sparse matrix multiplication requires $\mathcal{O}(nonZero(N))$. What's more, WWW is found to follow the power-law degree distribution [2], which means that the average web pages have 10 links [4], and $nonZero(M) \approx 10N$. Therefore, the total time complexity is reduce to $\mathcal{O}(kN)$.

References.

[1] Krishna Bharat and Monika R Henzinger. "Improved algorithms for topic distillation in a hyperlinked environment". In: *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. 1998, pp. 104–111 (cit. on p. 8).

[2] Bernardo A Huberman and Lada A Adamic. "Growth dynamics of the world-wide web". In: *Nature* 401.6749 (1999), pp. 131–131 (cit. on p. 8).

[3] Google Inc. *Google Press Center: Fun Facts*. July 2001. URL: https://www.google.com/competition/howgooglesearchworks.html (cit. on p. 7).

[4] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters". In: *Internet Mathematics* 6.1 (2009), pp. 29–123 (cit. on p. 8).

0.5 SSD: Single Shot MultiBox Detector

- *Algorithm:* SSD: Single Shot MultiBox Detector

- *Input:* A RGB 3 channel picture
- *Complexity:* None
- *Data structure compatibility:* None
- *Common applications:* Artificial intelligence

Problem. SSD: Single Shot MultiBox Detector

Object detection is the task of detecting instances of objects of a certain class within an image. A sample of object detection is shown in Figure 1.

Figure 1: **Detection examples on COCO `test-dev` with SSD512 model.** Each color corresponds to an object category.

SSD [2], a single-shot detector for multiple categories, is faster than the previous state-of-the-art for single shot detectors (YOLO [3]), and more accurate, region proposals and pooling (including Faster R-CNN [4]). The core of SSD is predicting category scores and box offsets for a fixed set of default bounding boxes using small convolutional filters applied to feature maps. To achieve high detection accuracy it produces predictions of different scales from feature maps of different scales, and explicitly separate predictions by aspect ratio. These design features further improve the speed vs accuracy trade-off by leading to simple end-to-end training and high accuracy, even on low resolution input images.

Description

Model

The SSD approach is based on a feed-forward convolutional network. Network produces a fixed-size collection of bounding boxes and scores for the object instances in those boxes. It then follows by a non-maximum suppression step to produce the final detections. The early network layers called the base network are based on a standard architecture used in image classification. It then add auxiliary structure to the network to produce detections with the following key features:

For multi-scale feature maps for detection, it adds convolutional feature layers to the end of the base network. These layers progressively decrease in size and predicts detections at multiple scales. The convolutional model for predicting detections is different for each feature layer.

For convolutional predictors for detection, each feature layer can use a set of convolutional filters to produce a fixed set of detection predictions . These are on top of the SSD network architecture in Fig. 2. For a $m \times n$ size feature layer with p channels, the basic element for predicting parameters of a potential detection is a $3 \times 3 \times p$ small kernel that produces either a score for a category, or a shape offset relative to the default box coordinates. It produces an output value at each of the $m \times n$ locations. The bounding box offset will be then measured.

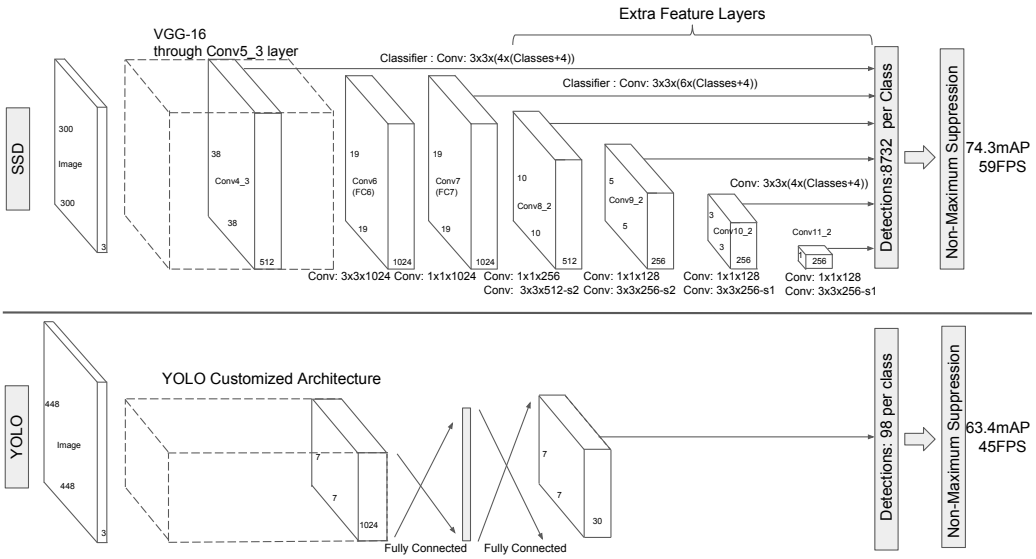

Figure 2: A comparison between two single shot detection models: SSD and YOLO [**redmon2015you**]. Several feature layers are added to the end of a base network in SSD, which predict the offsets to default boxes.

Training

The SSD training objective is inherited from the MultiBox objective [1]. But it should be extended to handle multiple object categories. Let $x_{ij}^p = \{1, 0\}$ be an indicator for matching the i-th default box to the j-th ground truth box of category p.

The matching strategy above illustrates $\sum_i x_{ij}^p \geq 1$. The overall objective loss function is a weighted sum of the localization loss (loc) and the confidence loss (conf):

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \tag{0.5.1}$$

where N is the number of matched default boxes. If $N = 0$, loss equals to 0. The localization loss is a Smooth L1 loss between the predicted box (l) and the ground truth box (g) parameters. The remaining is similar to Faster R-CNN [4].

$$\begin{aligned} L_{loc}(x, l, g) &= \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m) \\ \hat{g}_j^{cx} &= (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h \\ \hat{g}_j^w &= \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right) \end{aligned} \tag{0.5.2}$$

The confidence loss is the softmax loss over multiple classes confidences (c).

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)} \tag{0.5.3}$$

and the weight term α is set to 1 by cross validation.

References.

[1] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. "Scalable object detection using deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 2147–2154 (cit. on p. 10).

[2] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37 (cit. on p. 9).

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788 (cit. on p. 9).

[4] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99 (cit. on pp. 9, 10).

0.6 YOLO: You Only Look Once

- *Algorithm:* YOLO: You Only Look Once
- *Input:* A RGB 3 channel picture
- *Complexity:* None
- *Data structure compatibility:* None
- *Common applications:* Artificial intelligence

Problem. YOLO: You Only Look Once

Object detection is the task of detecting instances of objects of a certain class within an image. A sample of object detection is shown in Figure 3. Prior detection systems [4] repurpose classifiers or localizers to perform

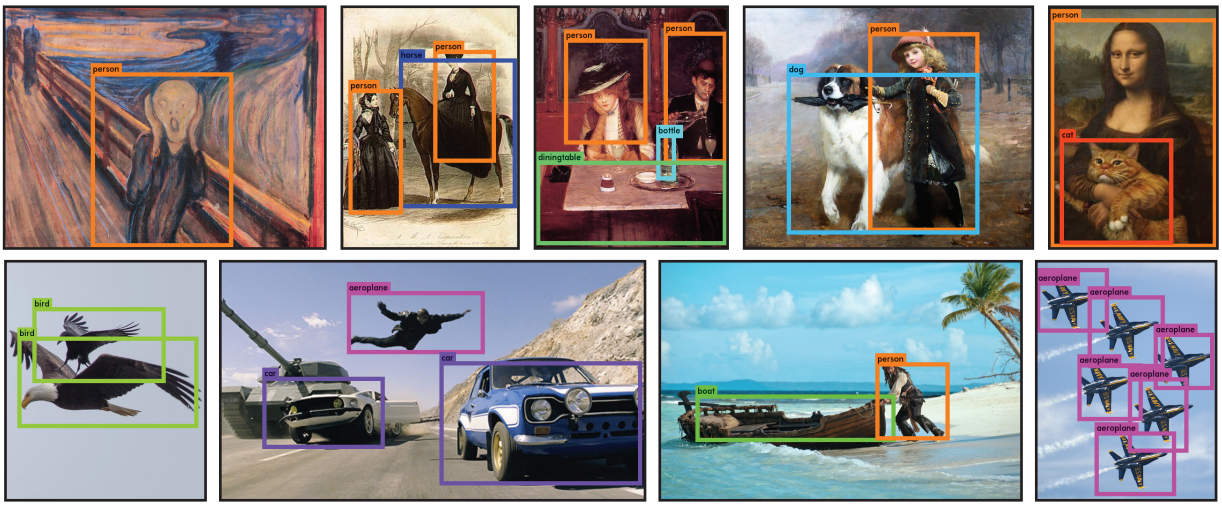

Figure 3: The result of object detection

detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

YOLO [3] uses a completely different approach where only a single neural network is applied to the full image. This network divides the image into grids and predicts bounding boxes and probabilities for each grid. These bounding boxes are weighted by the predicted probabilities.

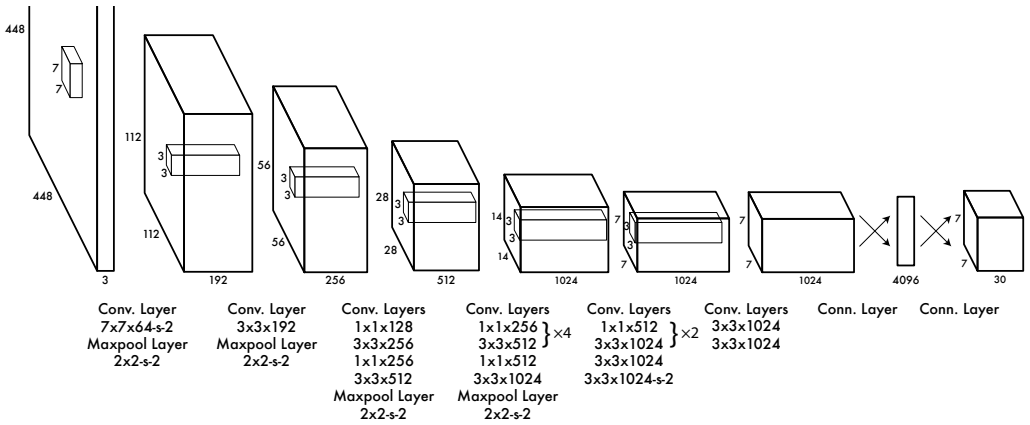

Figure 4: **The Architecture.** The detection network has 24 convolutional layers and 2 following fully connected layers. To be noted that 1×1 convolutional layers are alternated.

YOLO has several merits. It looks at the whole image at test time so its predictions are informed by global context in the image. It only uses a single network evaluation to predict unlike region-based network which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN [2] and 100x faster than Fast R-CNN [1].

Description

Detection Method

YOLO system cuts the input image into an $S \times S$ grid. If the center of an object is in a grid cell, that grid cell will detect that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores is an value of how confident and accurate is that the box contains an object.

Network

The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates.

The network architecture follows the GoogLeNet model for image classification [6]. The network has 24 convolutional layers followed by 2 fully connected layers. It alternatively use 1×1 reduction layers followed by 3×3 convolutional layers. Figure 4 shows the network.

Training

It pretrains its convolutional layers on the ImageNet 1000-class competition dataset [5]. For pretraining, only first 20 convolutional layers from Figure 4 are used followed by a average-pooling layer and a fully connected layer.

It then converts the model to perform detection. It adds four convolutional layers and two fully connected layers. They are randomly initialized weights. They increases the input resolution of the network from 224×224 to 448×448. The final layer predicts both class probabilities and bounding box coordinates. It falls between 0 and 1 by normalizing the bounding box width and height by the image width and height so that they .

Leaky rectified linear activation function are used for the final layer and all other layers:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \tag{0.6.1}$$

YOLO can predict multiple bounding boxes in each grid. But it only needs one bounding box predictor to be responsible for each object at training time. one predictor well be responsible for predicting an object. The assignment is based on which prediction has the best IOU with the ground truth.
The following multi-part loss function will be used in training:

$$
\begin{aligned}
&\lambda_{\mathbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \infty_{ij}^{\mathrm{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\mathbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \infty_{ij}^{\mathrm{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \infty_{ij}^{\mathrm{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\mathrm{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \infty_{ij}^{\mathrm{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \infty_{i}^{\mathrm{obj}} \sum_{c \in \mathrm{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

where ∞_i^{obj} denotes if object appears in cell i and $\infty_{ij}^{\mathrm{obj}}$ denotes that the jth bounding box predictor in cell i is responsible for that prediction.

To avoid overfitting it uses dropout and extensive data augmentation. Drop out rate is 0.5. For data augmentation, it composes random scaling and translations. Exposure and saturation will be also randomly adjusted in the HSV color space.

References.

[1] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448 (cit. on p. 12).

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Region-based convolutional networks for accurate object detection and segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 38.1 (2015), pp. 142–158 (cit. on p. 12).

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788 (cit. on p. 11).

[4] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99 (cit. on p. 11).

[5] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252 (cit. on p. 12).

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015 (cit. on p. 12).