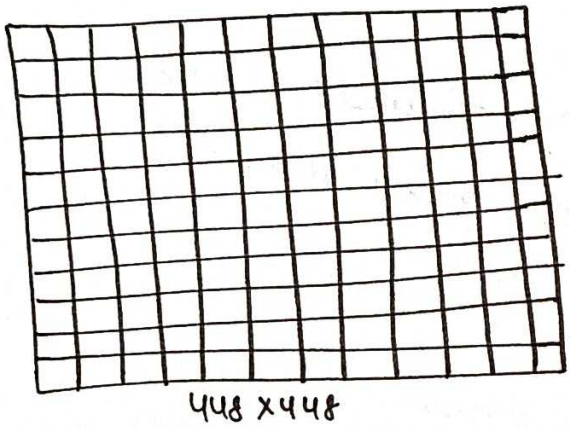
YOLO (You Only Look Once)

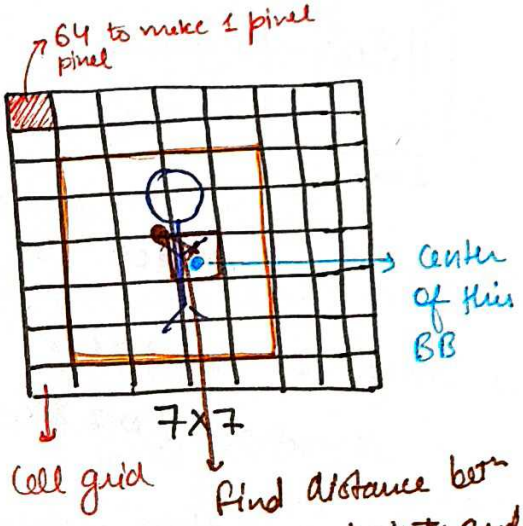
YOLO → Real time of detection
↳ one shot → regression
↳ losses (obj, BB, classif)

Step-1:

Resize the image to 448 x 448

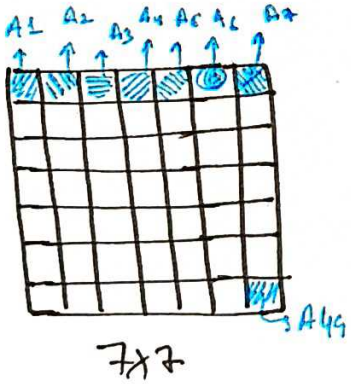


convert



$\frac{448}{7} = 64$

Image → 448 x 448
↳ convert into 7x7

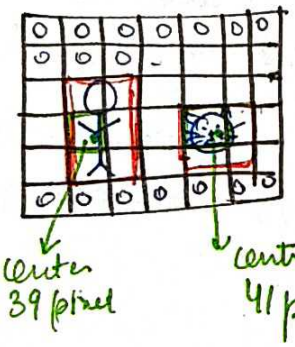


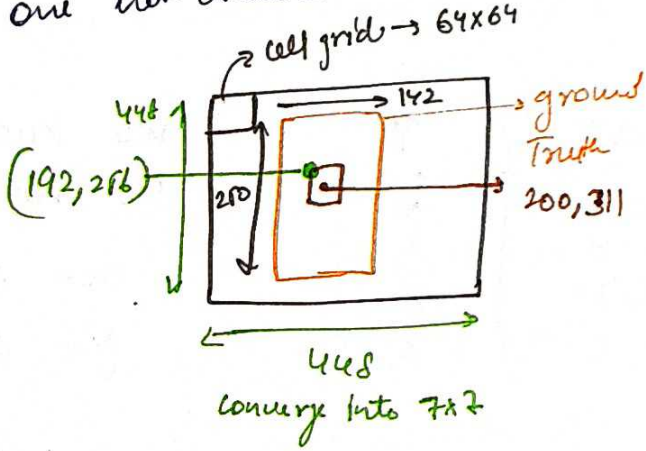
7x7
↓
A_1
A_2
A_3
A_4
⋮
A_{49}

	Δx	Δy	Δw	Δh	Ci(objectscore)	$\vec{P}$(20 Class) (0000…0)+1 Grid 20
A_1	0	0	0	0	0	0 0 0 0 …
A_2	0	0	0	0	0	0 — —
⋮						Person
				0.5	1	[0 0 0 1 0 …0]
		0.7	0.1			Last
A39	0.31			0.7	1	[0 1 0 0 0 0]
A40	0.4	0.8	0.6			
⋮						[0 0 0 0 0 0]
A_{49}	0	0	0	0	0	

Chart ↑

[$\vec{P}$(20 class) = Pascal data with 20 classes and all classes are one hot encoded]





$\Delta w = 142/448 = w_1$

$\Delta h = 250/448 = h_1$

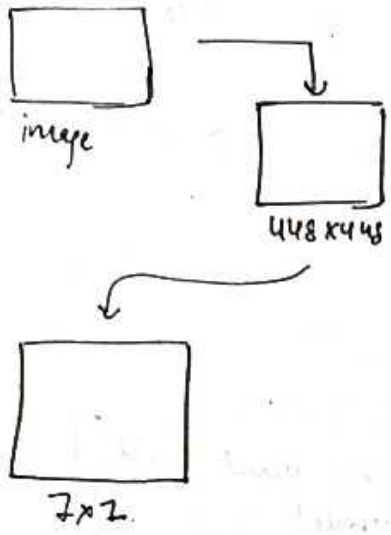
$\Delta x = \frac{200 - 192}{64} = x_1$

$\Delta y = \frac{311 - 256}{64} = y_1$

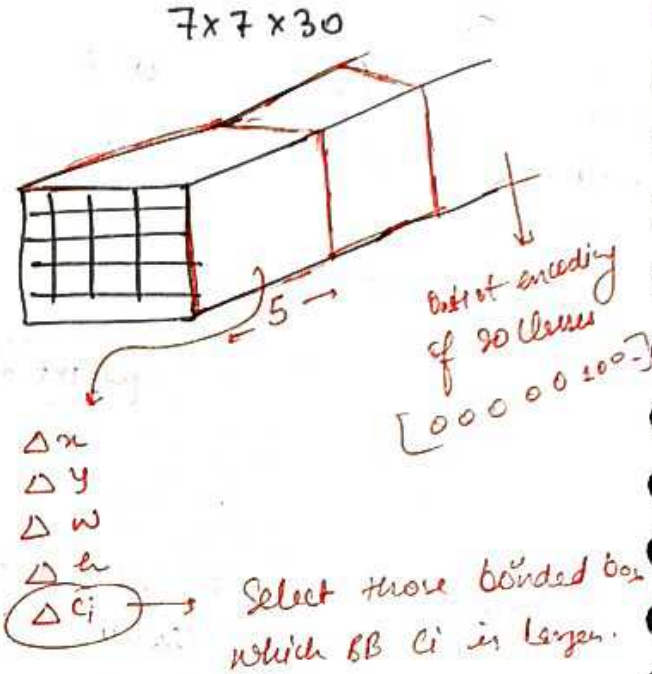
* These values we in upper chart.

Architecture

Training



Prediction



Δx, Δy → x, y
Δw → w
Δh → h

We know Δx, Δy, Δw and Δh
So, we can easily find out x, y, w, h

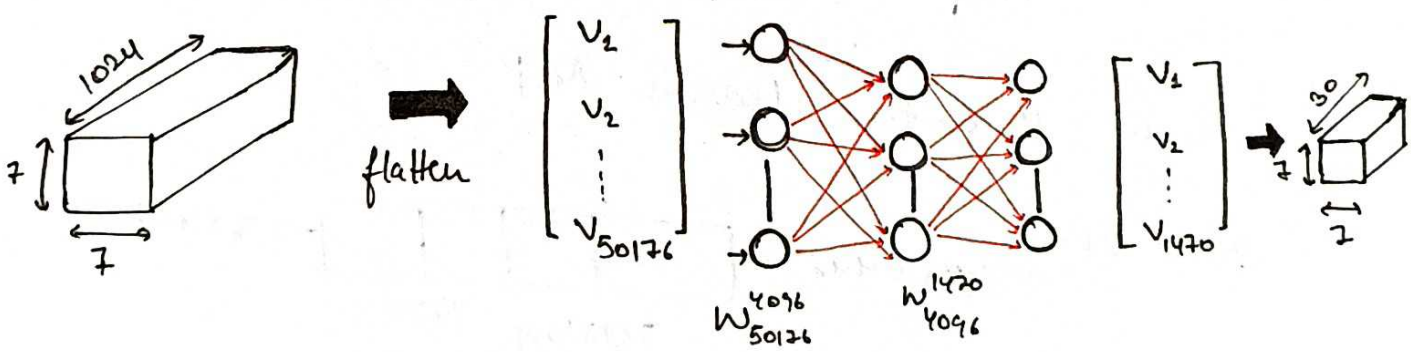
$x = \Delta x + 64 + (192 - 200)$, $w = \Delta w \times 448$
$y = \Delta y + 64 + (256 - 311)$, $h = \Delta h \times 448$

(x, y, h, w) → Grid cell Top left corner

Structure

Type	Size	Filter	Stride	output
conv.	7x7x3	64	2	224 x 224 x 64
max pool	2x2			112 x 112 x 64
conv.	3x 3x 64	192	1	112 x 112 x 192
maxpool	2x2			56 x 56 x 192

Conv.	1×1×192	128	1	56×56×128
Conv.	3×3×128	256	1	56×56×256
Conv.	1×1×256	256	1	56×56×256
Conv.	3×3×256	512	1	56×56×512
maxpool	2×2			28×28×512
4× Conv.	1×1×512	256	1	28×28×256
4× Conv.	3×3×256	512	1	28×28×512
Conv.	1×1×512	512	1	28×28×512
Conv.	3×3×512	1024	1	28×28×1024
maxpool	2×2			14×14×1024
2× Conv.	1×1×1024	512	1	14×14×512
2× Conv.	3×3×512	1024	1	14×14×1024
Conv.	3×3×1024	1024	1	14×14×1024
Conv.	3×3×1024	1024	2	14×14×1024
Conv.	3×3×1024	1024	1	14×14×1024
Conv.	3×3×1024	1024	1	14×14×1024



YOLO V2

YOLO V2 → Accuracy
→ Increase No. of Objects
→ Increase Speed

Modification and Improvement in YOLO V2 over YOLO V1

1. MaP grew from 64 to 78.

(A) Apply Batch Normalization to all CNN layers.
→ improve map by 2%.
→ Solved overfitting
→ regularization effects

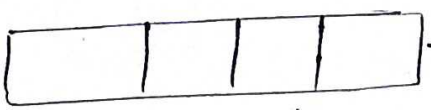
(B) Higer Resolution classifier

YOLO V1 → Pretrained (224 × 224) → Train on → detection (448 × 448)

YOLO V2 → Pretrained (224 × 224) → Train on (convert) (448 × 448) → detection (448 × 448)

→ Improve MaP by 4%.

(C) High Resolution feature MaP

YOLO V1 → 448 × 448 →  → 7×7×30

7×7×1024 1470

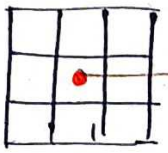
YOLOV2 →

448 x 448 → 14 x 14 x 1024

Remove Max pool from YOLO V1

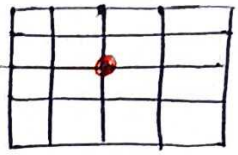
but if we do even x even x 1024 this is not good

We can do odd x odd



Solution ← Here's perfect Center

This is centre and we don't want this centre

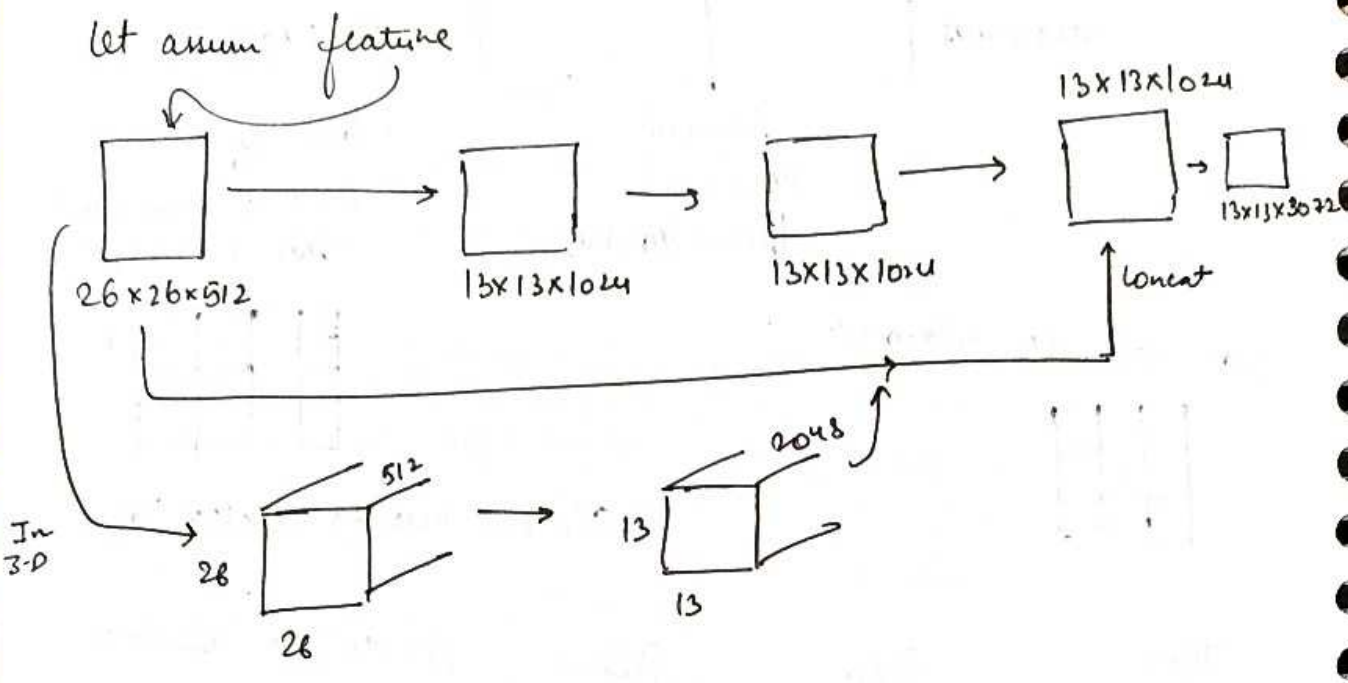


So, we use → 13 x 13 x 1024

Type	Size	Filters	Stride	outputs
		Same as YOLO V1		
4x [Conv.	1 x 1 x 512	256	1	28 x 28 x 256
Conv.]	3 x 3 x 256	512	1	28 x 28 x 512
Conv.	1 x 1 x 512	512	1	28 x 28 x 512
Conv.	3 x 3 x 512	1024	1	28 x 28 x 1024
max pool		Remove in YOLO V2		
2x [Conv.	1 x 1 x 1024	512	1	28 x 28 x 512
Conv.]	3 x 3 x 512	1024	1	28 x 28 x 1024
Conv.	3 x 3 x 1024	1024	1	28 x 28 x 1024
Conv.	3 x 3 x 1024	1024	2	14 x 14 x 1024
Conv.	3 x 3 x 1024	1024	1	14 x 14 x 1024
Conv.	3 x 3 x 1024	1024	1	14 x 14 x 1024

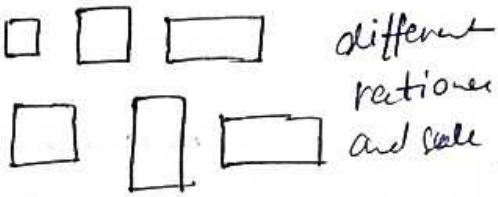
These values are changed

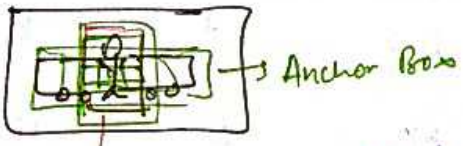
④ Add Pass Through Layer



⑤ Bounding Box Improvement

→ Use Anchor Box →



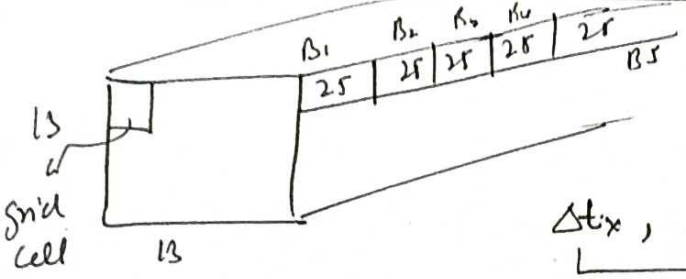


↳ YoloV2 detect person but hard to detect bus. So Anchor Box use to detect different shape as well.

→ In Yolo Faster R-CNN → We decide No. of Anchor Box shape

→ In YoloV2 → K-Mean Cluster decide different Anchor Box.

5 Diff Modification which improve the model drastically from map of 64% to 78%.



→ 5 Achor Box for every grid cell

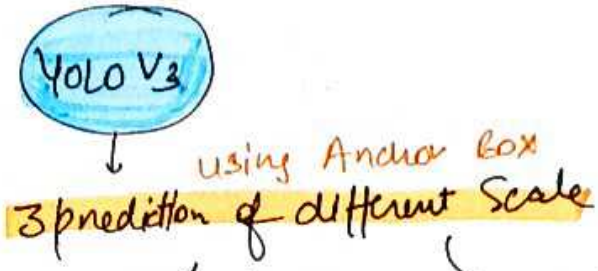
Δt_x, Δt_y, Δt_w, Δt_h, 20 classes

25

Total Parameter = 125 → 1 grid cell

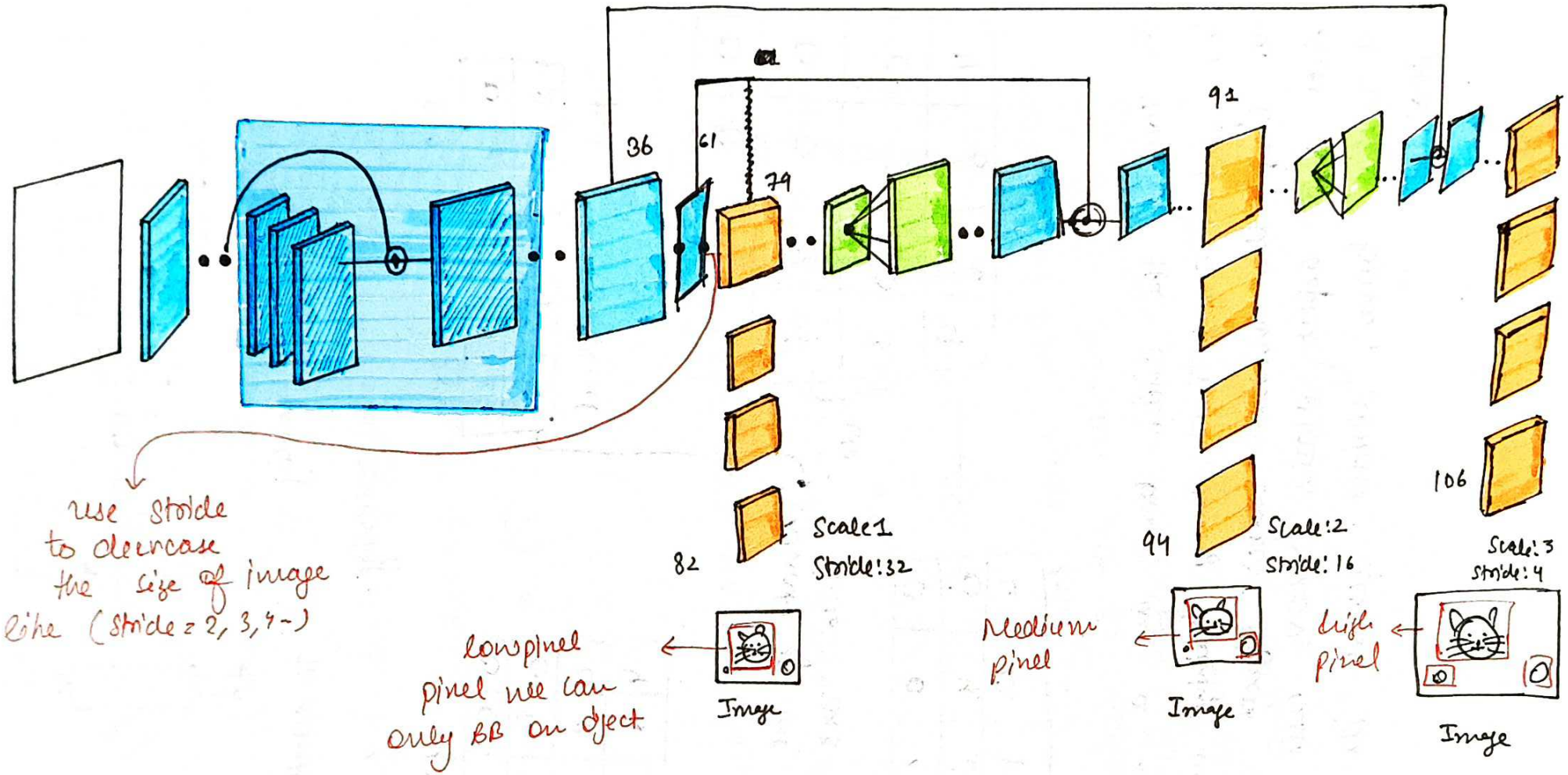
YOLO V2 Parameter = 13 × 13 × 125

YOLO V3

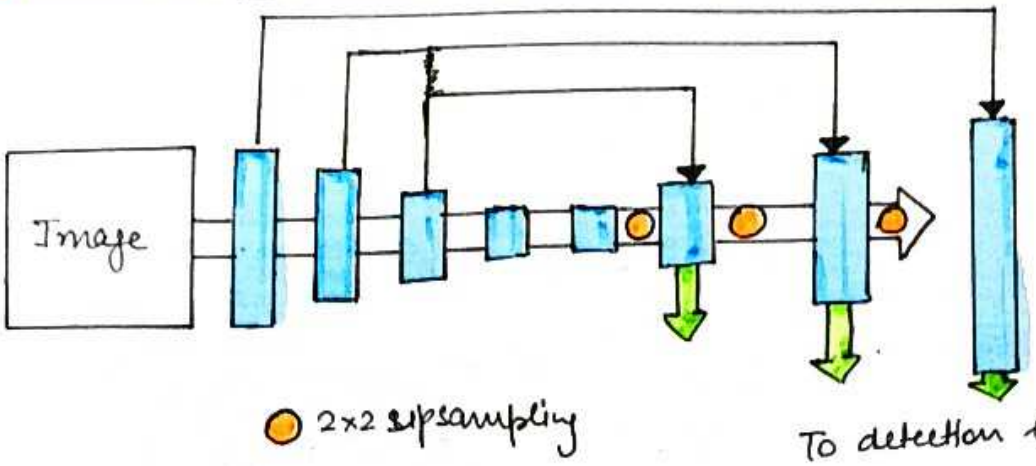


In YOLO V_2 architechture use → DarkNet (19) → 19 Layer

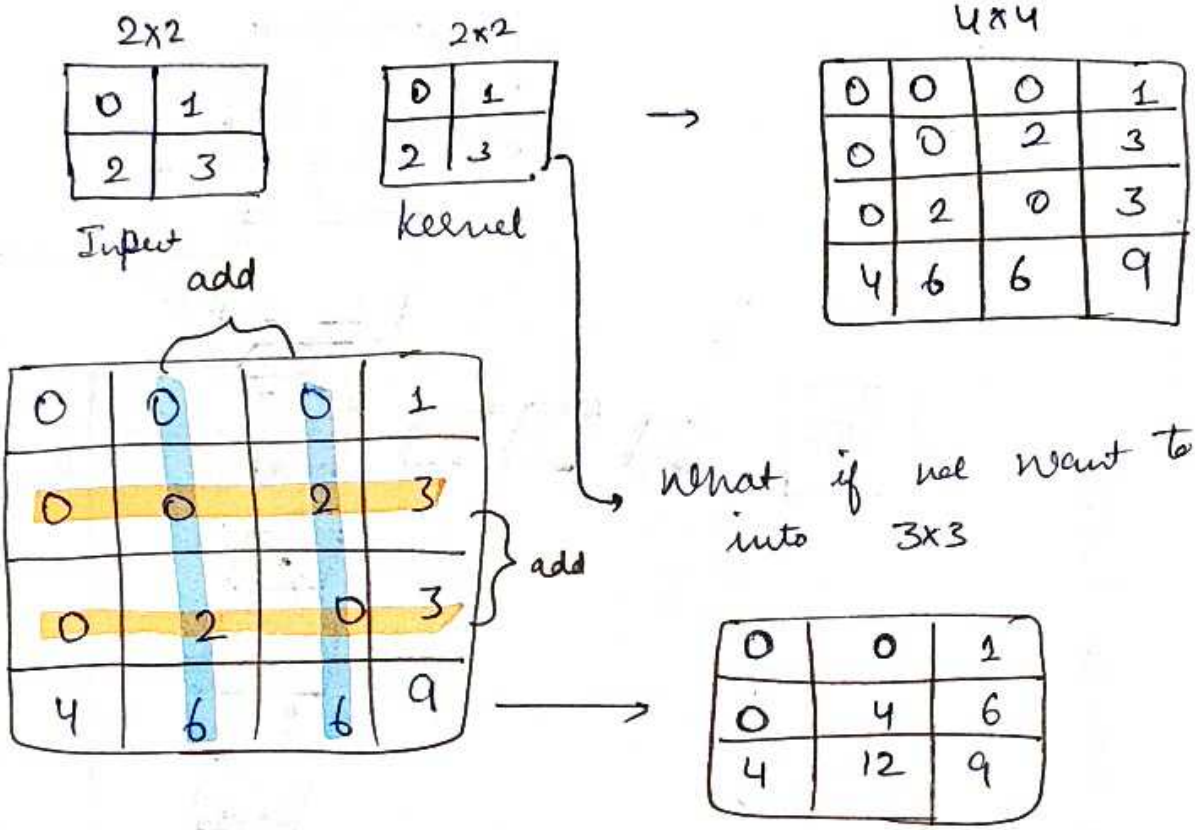
In YOLO V_3 architechture use → DarkNet (53) → 53 Layer



YOLO V_3 network Architecture



Upsampling



In YOLOv3, the anchor boxes are a set of predefined bounding box shapes used during training and prediction to help the model detect objects of various sizes and aspect ratios YOLOv3 uses a total of 9 anchor boxes, divided across its three different scales.

The anchor box sizes in YOLOV3 are as follows:

For the first scale (largest scale):

Anchor box 1: Width = 10, Height = 13

Anchor box 2: Width = 16, Height = 30

Anchor box 3: Width = 33, Height = 23

For the second scale (Medium scale)

Anchor box 1: width = 30, Height = 61

" " 2: " = 62, " = 45

" " 3: " = 59, " = 119

For the Third Scale (Smallest scale):

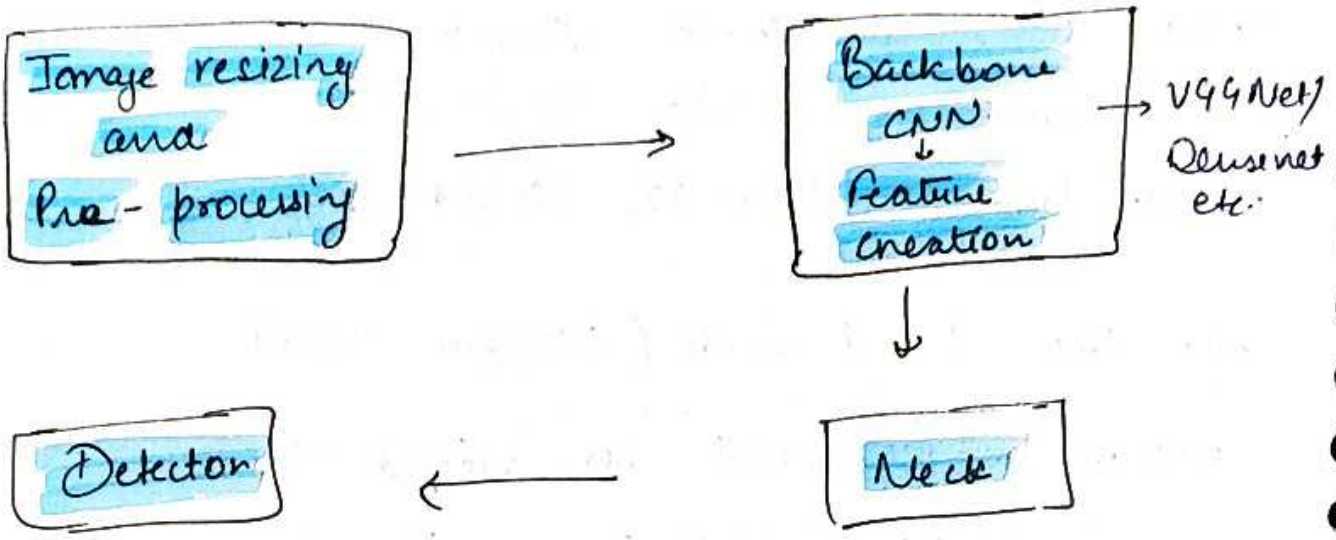
Anchor box 1: Width = 116, Height = 90

" " 2: " = 156, " = 198

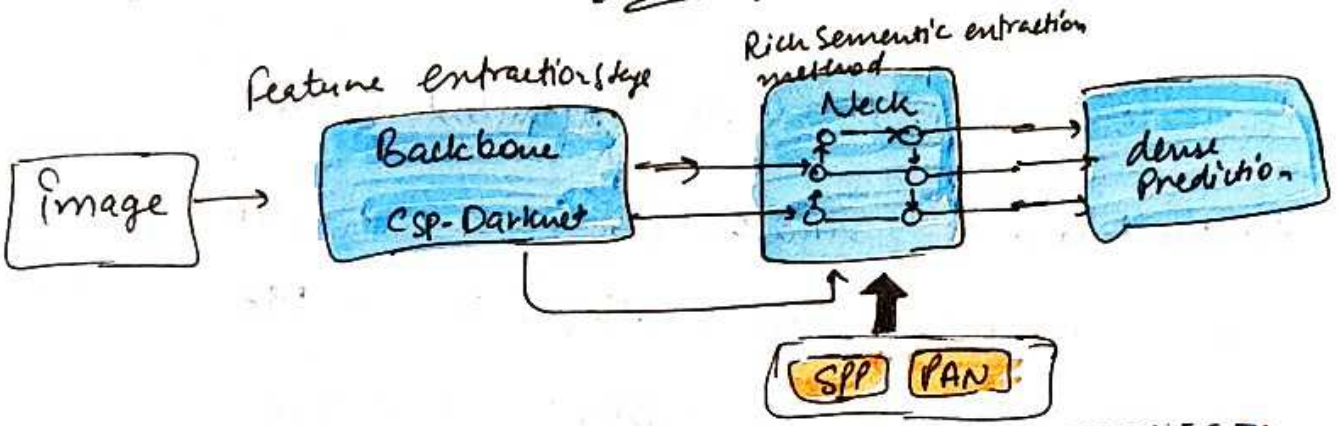
" " 3: " = 373, " = 326

These anchor box sizes were determine based on the analysis of the dataset used for training YOLOV3. They cover range of object commonly founded in dataset. and model learn to adjust the prediction on the based of anchor box during training.

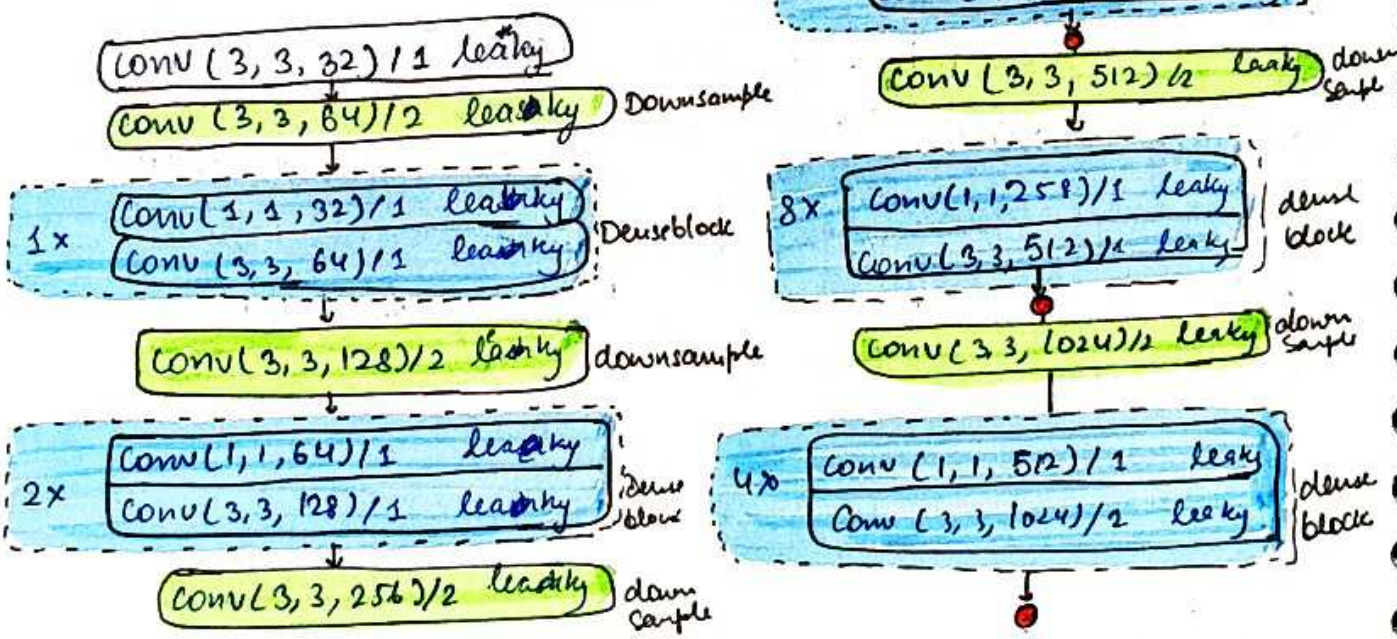
Object detection Rule for any architecture



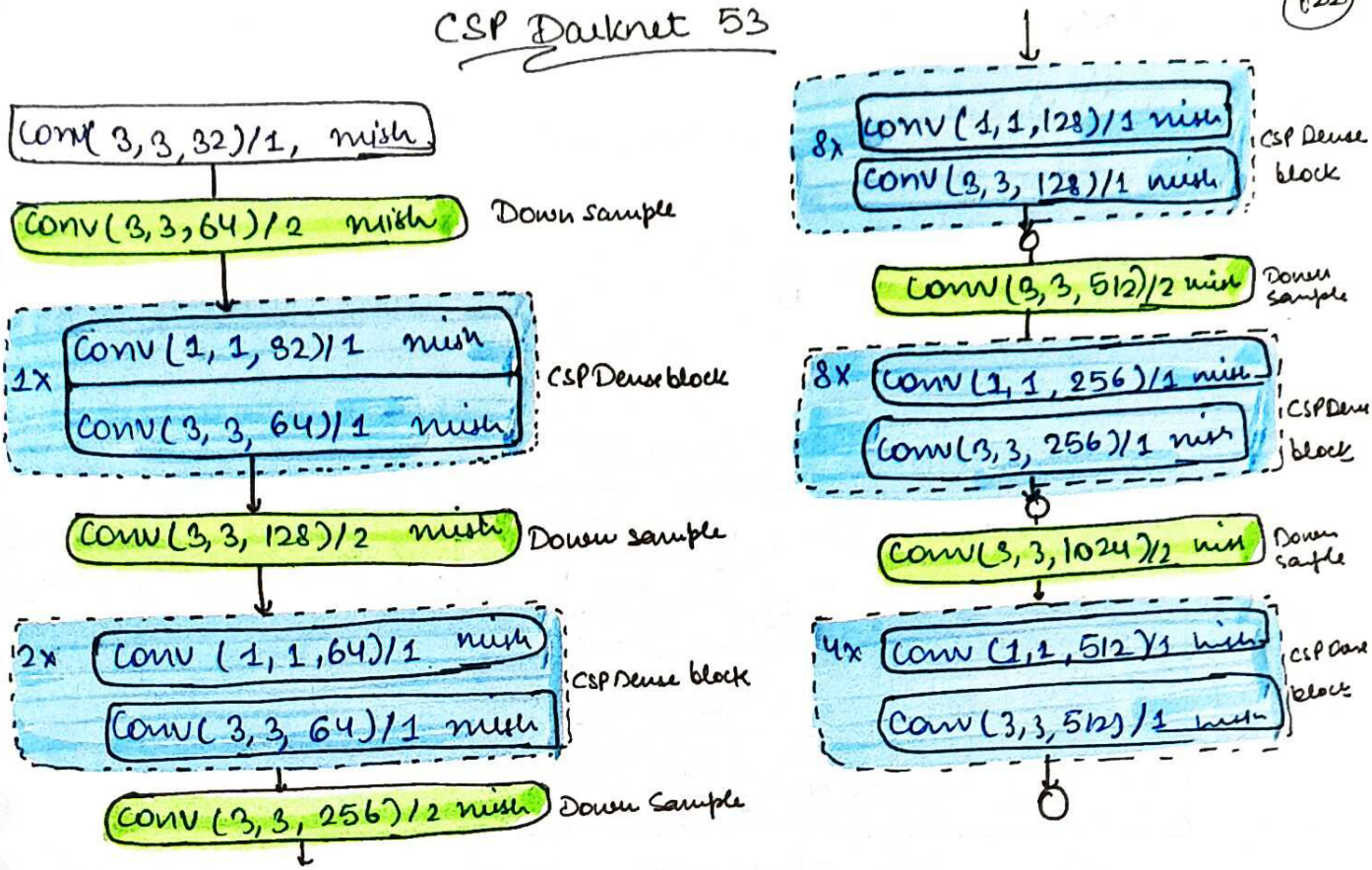
YOLO V4



Darknet 53



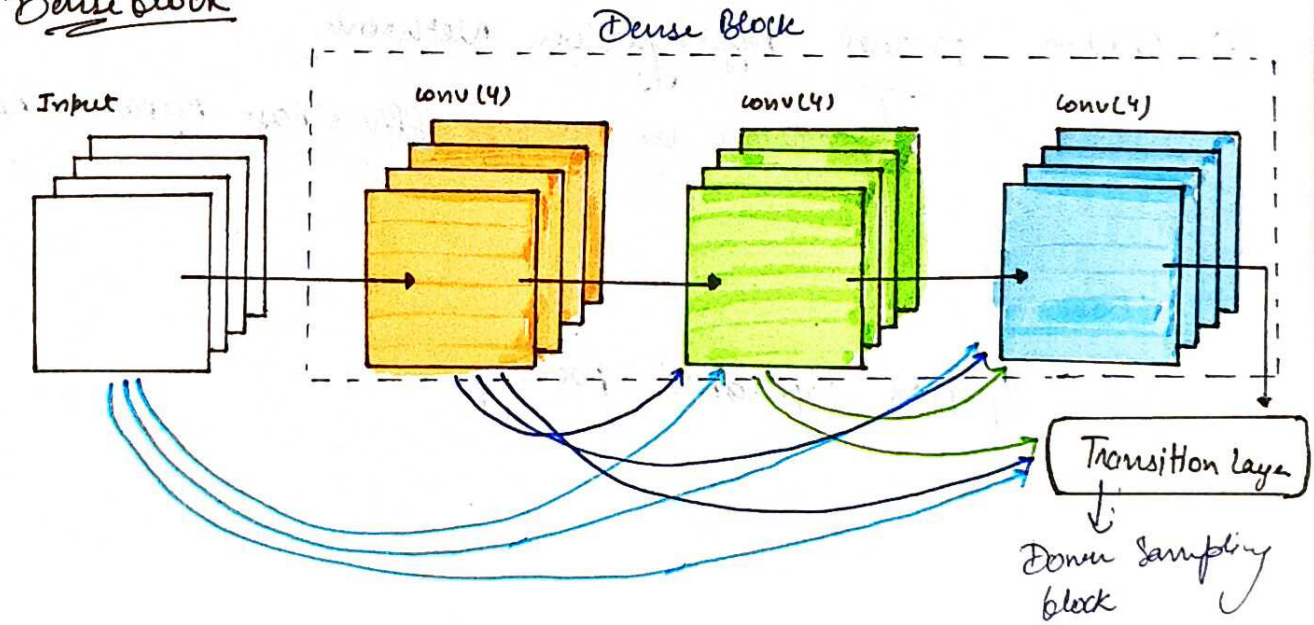
CSP Darknet 53



Difference betn Darknet and CSP Darknet

1. Activation function change (LeakyRelu → MISH)
2. Dense block → CSP-DenseBlock

Dense block

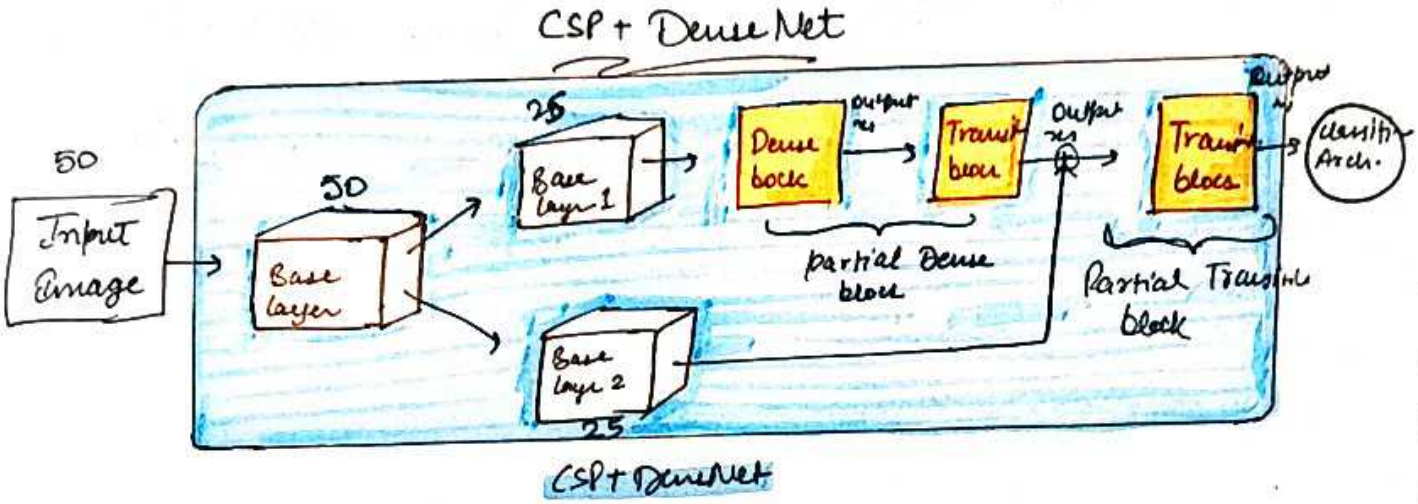


$x_1 = w_1 * x_0$

$x_2 = w_2 * [x_0 x_1]$

⋮

$x_K = w_K * [x_0, x_1, \ldots - x_{K-1}]$



In YoloV4 → Changes → (i) BackBone CNN → CSP + Darknet

↓

Dense Block

+

Transition Block

(ii) Neck → SPP + PAN

We use typically → 2 feature Aggregation in YOLO V4

① PAN → Partial Aggregation Network

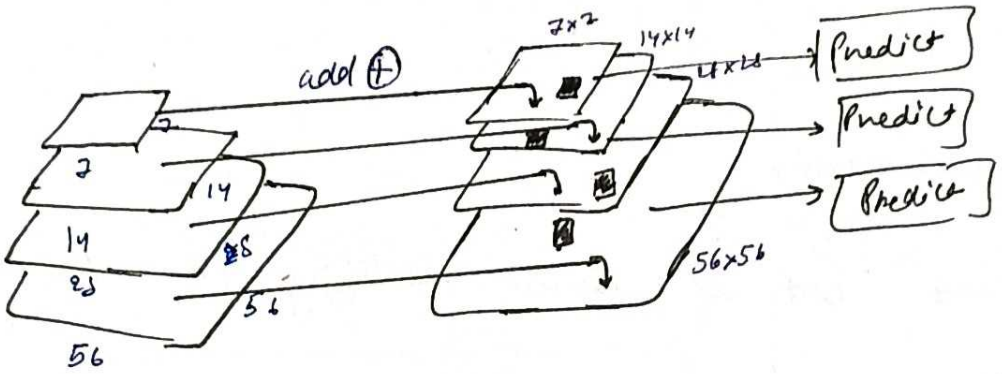
↳ advance version of FPN → Feat Pyramid Network

② SPP

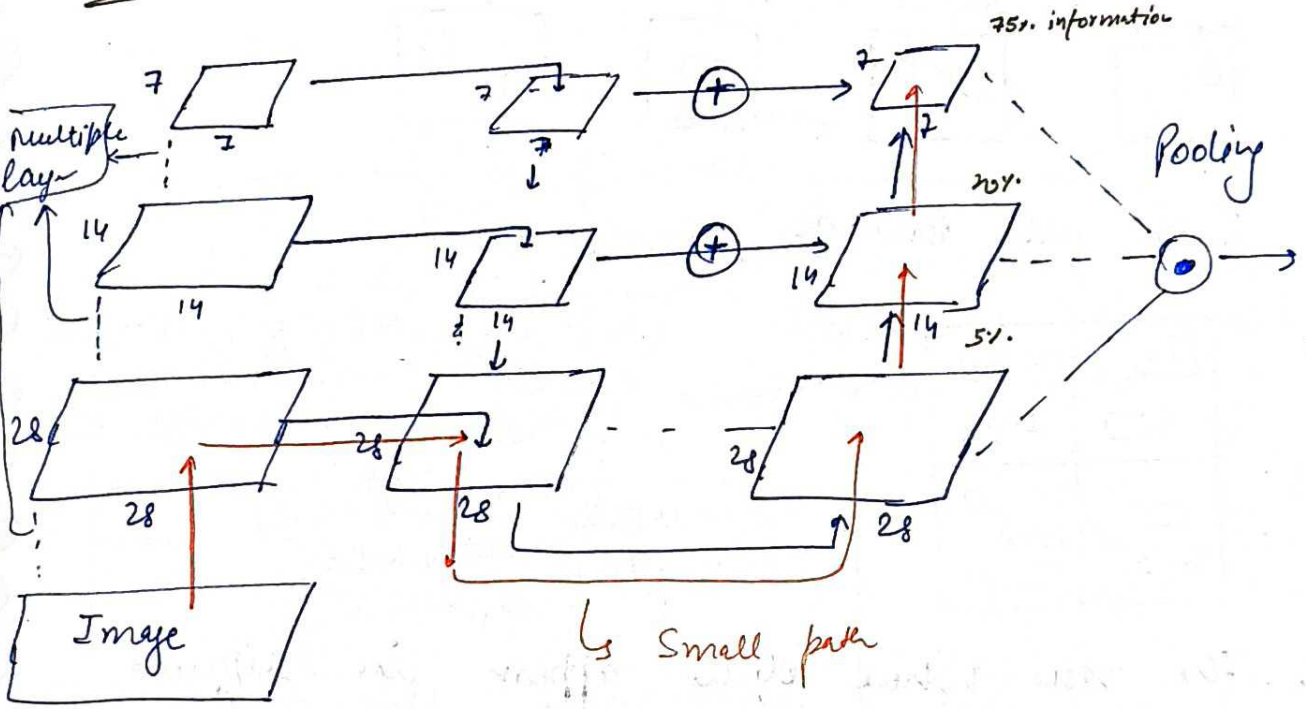
↳ Spatial Pyramid Pooling

FPN → Feature Pyramid Network

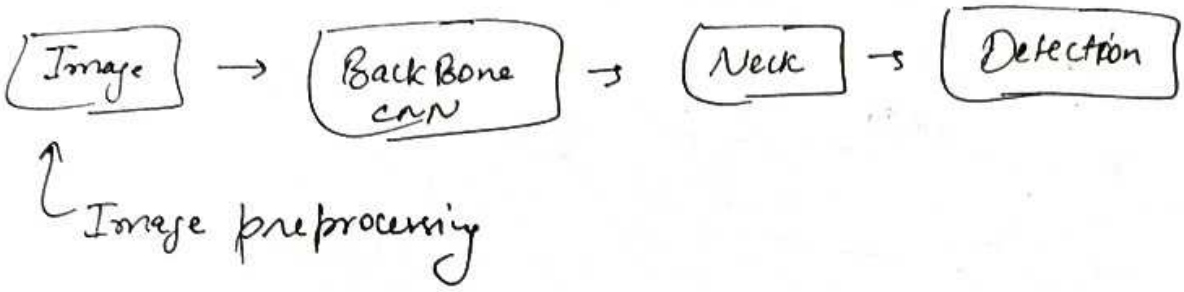
↳ Used to detect smallere object.



PAN → Path Aggregation Network



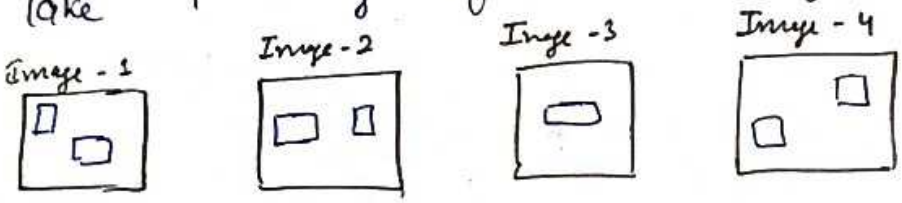
YOLO V5



① MOSAIC → Out of Content Identifying (Data Augmentation)

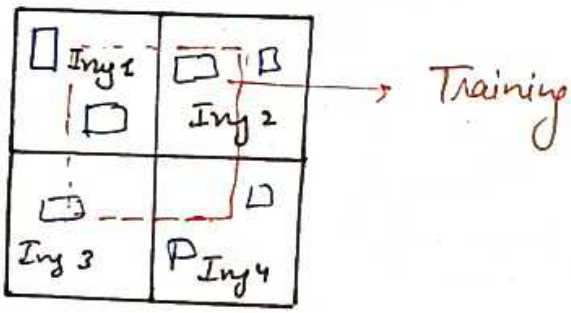
Steps:

(i) Take 4 image from the trainig set.



(ii) Resize all Image

(iii)



Mosaic will help the model to recognise the objects that the models is not used to looking together.

All the cases where objects appear in different content.

MOSAIC used only Trainig Time.

→ Bag of Freebies

↓

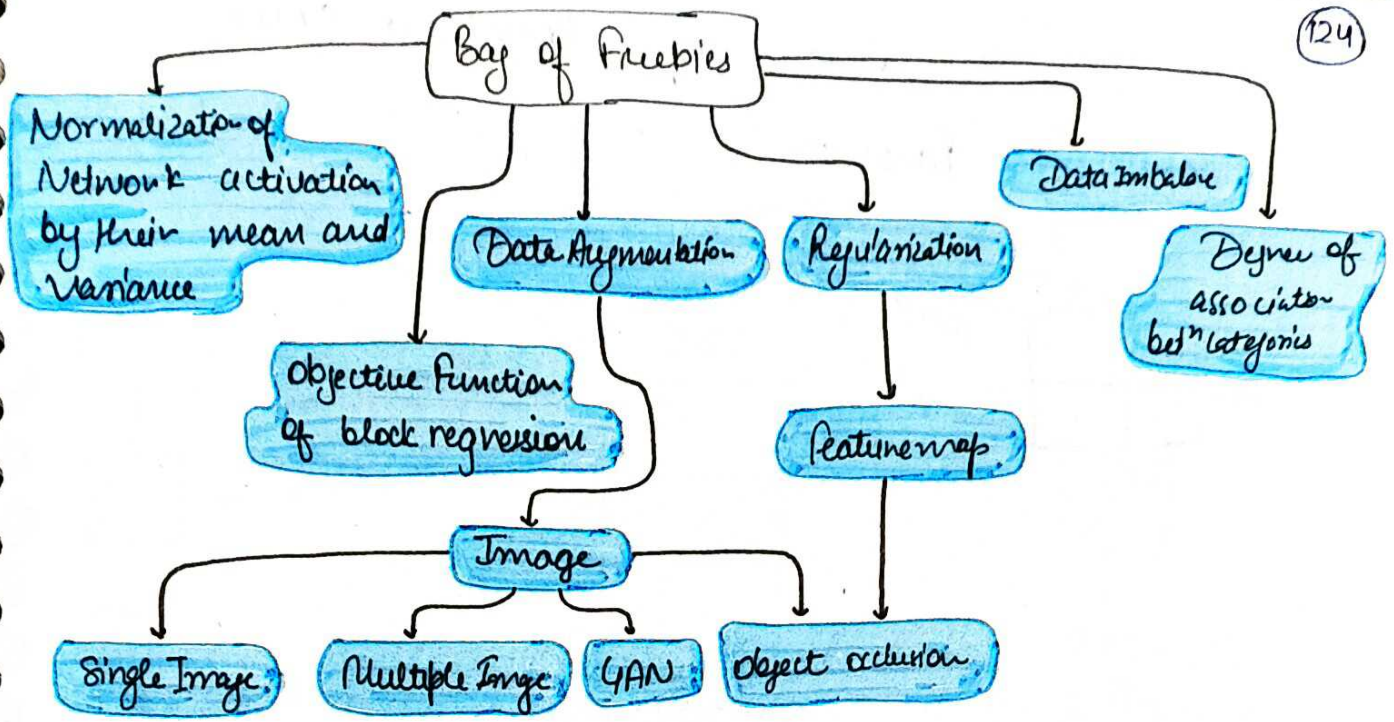
Traing Time ↑↑

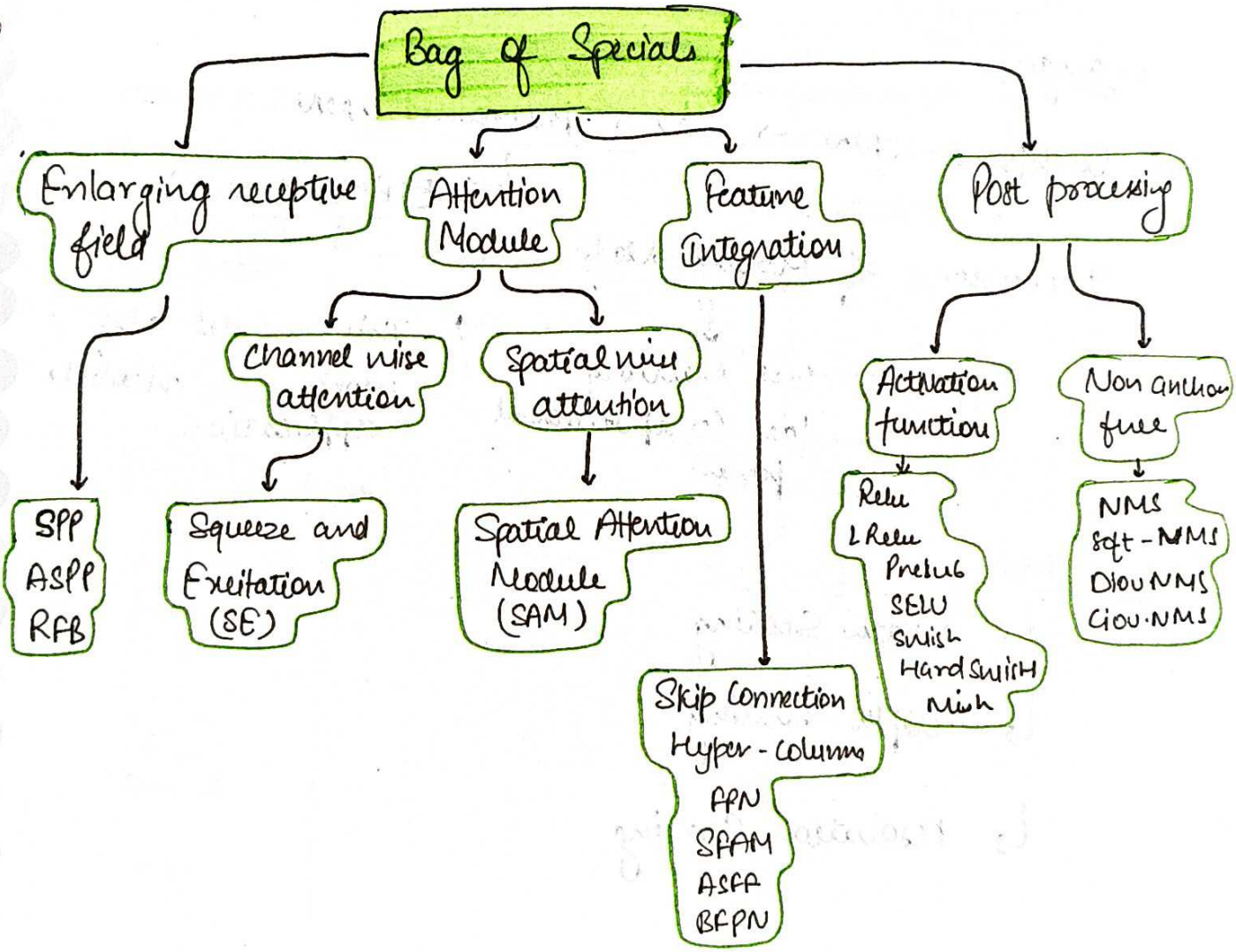
(Mosai data augmentation)

→ Bag of Specials

Architecture Major changes

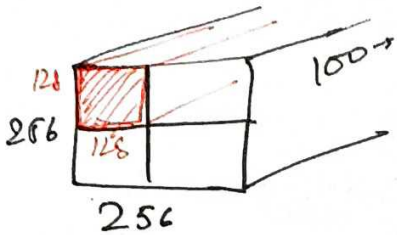
* Increase complexity also Increase Accuracy

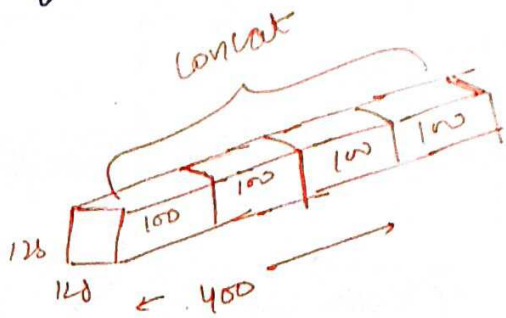

Bag of Freebies
Normalization of Network activation by their mean and variance
Data Augmentation
Regularization
Data Imbalance
Degree of association bet^n categories
Objective Function of block regression
Feature map
Image
Single Image
Multiple Image
GAN
Object occlusion


Bag of Specials
Enlarging receptive field
Attention Module
Feature Integration
Post processing
Channel wise attention
Spatial wise attention
Activation function
Non anchor free
SPP
ASPP
RFB
Squeeze and Excitation (SE)
Spatial Attention Module (SAM)
Relu
LRelu
Prelu6
SELU
Swish
HardSwish
Mish
NMS
Soft-NMS
DIoU NMS
CIoU-NMS
Skip Connection
Hyper-columns
FPN
SFAM
ASFF
BFPN

Focus Layer → Just before backbone CNN

↳ Reduce the Parameter





YOLO V6

Map → 2.5% and Speed → 2.5× faster

Changes

1. BackBone CNN → New pretrained CNN

↳ EfficientNET

Successor of Mobile Net ←

↓

less Memory
less computational power

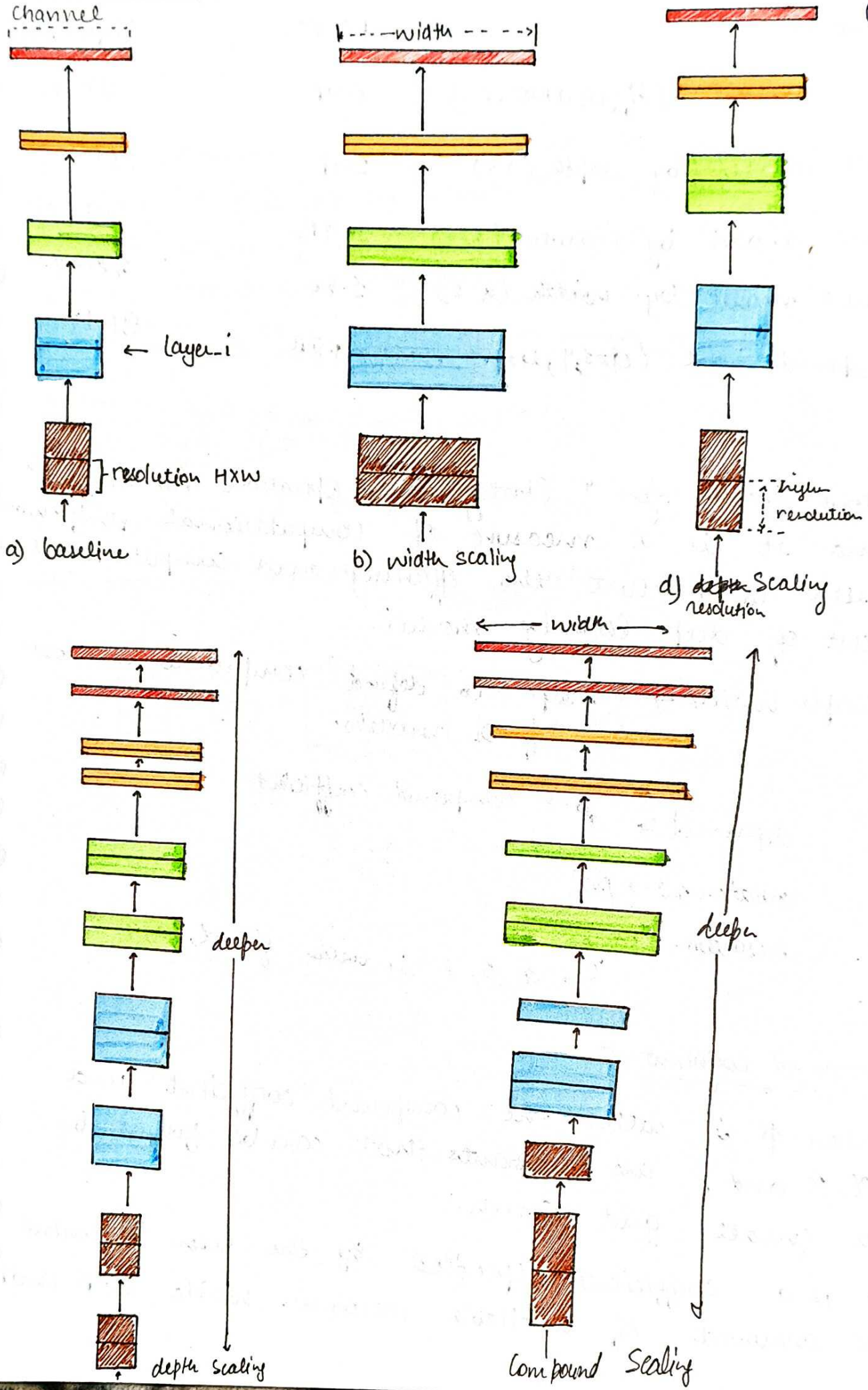
→ This model also work on mobile application.

Efficient Net

↳ Width Scaling

↳ depth Scaling

↳ resolution Scaling


channel
layer_i
resolution HxW
a) baseline
width
b) width scaling
higher resolution
d) depth Scaling
resolution
deeper
depth scaling
width
deeper
Compound Scaling

Models	FLOPs	Top1 Accuracy
Baseline model (EfficientNet-B0)	0.4B	77.3%
Scale model by depth (d=4)	1.8B	79.0%
Scale model by resolution (r=2)	1.9B	79.1%
Scale model by width (w=2)	1.8B	78.9%
Compound Scale (d=1.4, w=1.2, r=1.3)	1.8B	81.1%

↳ Less parameter and high accuracy

Flops stands for "Floating-point Operations Per Second" and it is a measure of computational performance used to evaluate the efficiency and computational cost of deep learning models.

Simple words → helps to define computational cost of DL models.

depth = d → α^{ϕ} → compound coefficient

width = w → $\beta\phi$

resolution = r = γ^{ϕ}

↳ α, β, γ is value of d, w, r

Compound Coefficient ϕ

Where ϕ is called the compound coefficient and α, β and γ are constants that can be found by a small grid search.

ϕ is a coefficient specified by the user to control the amount of available resources. while α, β and

assign these resources to the network's depth, width and resolution respectively.

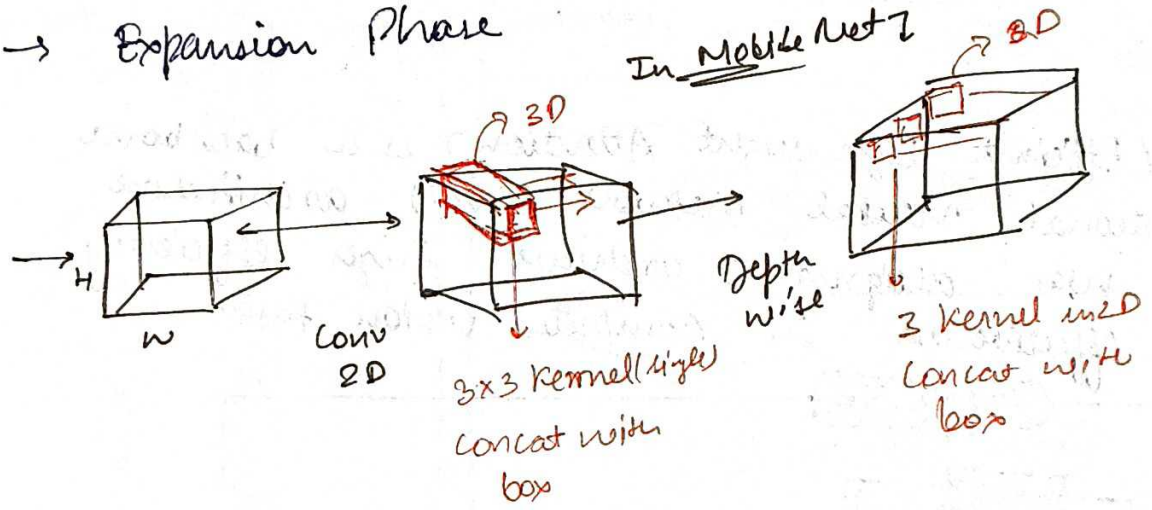
In simple word → We don't want to change (α, β and γ). Because α, β and γ is constant. If we want to change value of α, β and γ. We can easily change the of compound coefficient (ϕ).

eg → α^2, β^2, γ^2

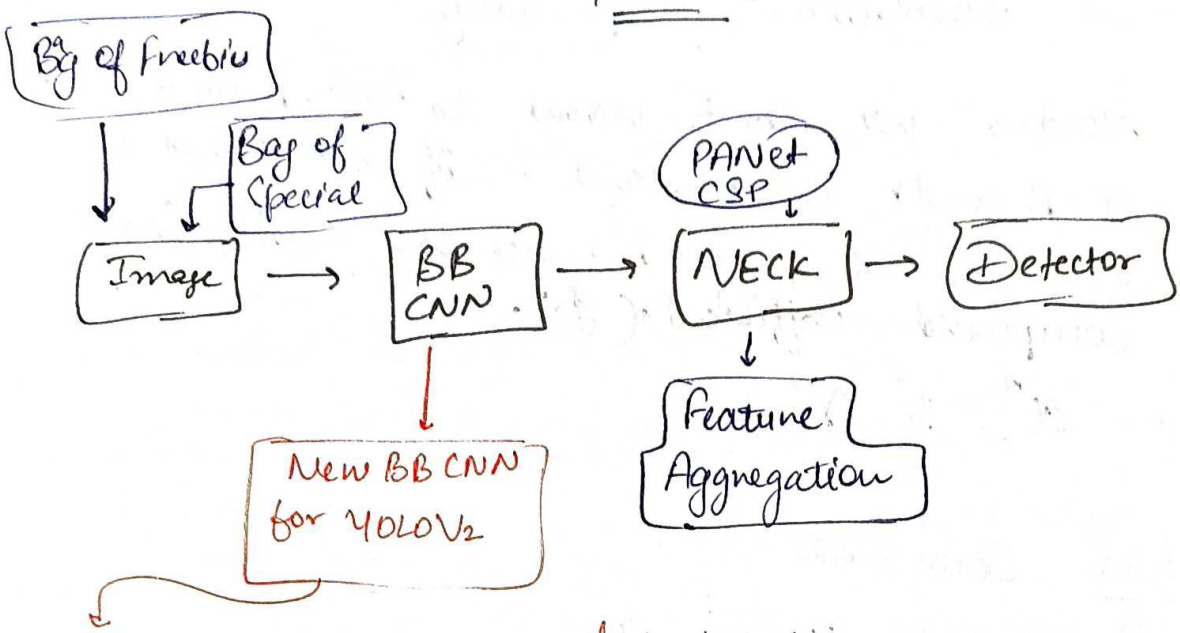
Mobile Net Conv Block

→ Squeeze and Excitation Phase

→ Expansion Phase



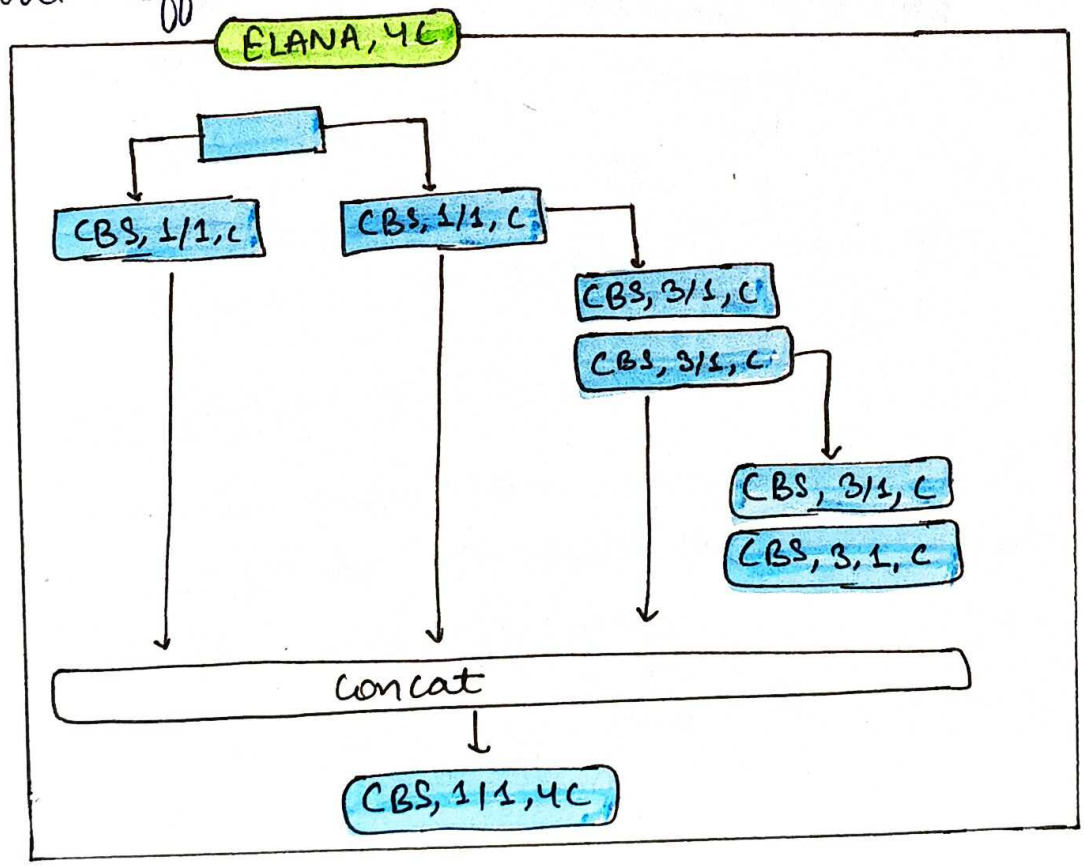
YOLO V7

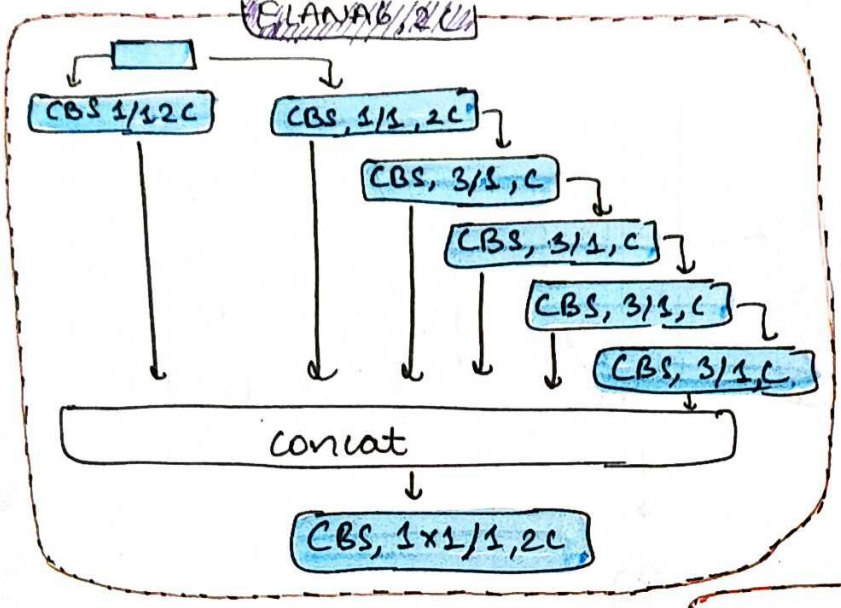


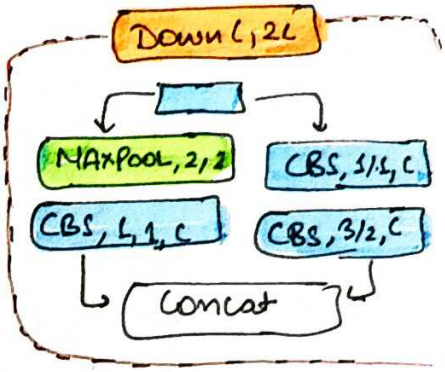
Fast and Accurate than efficient Net

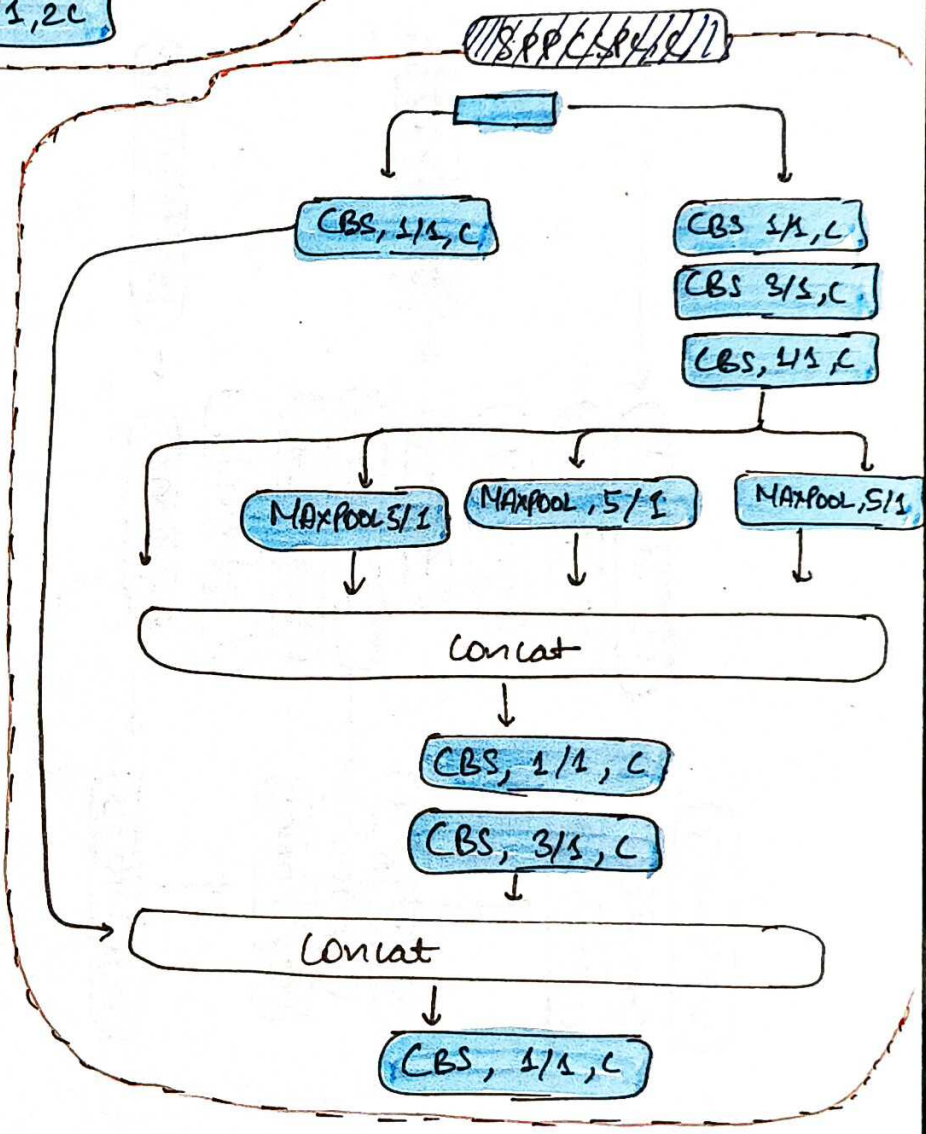
↳ Using Mechanism Attention → Ignore irrelevant things or information.

ELAN (Efficient Lightweight Attention) is a backbone convolutional neural network (CNN) architecture that was designed to archieve high efficiency and effectiveness in computer vision task.

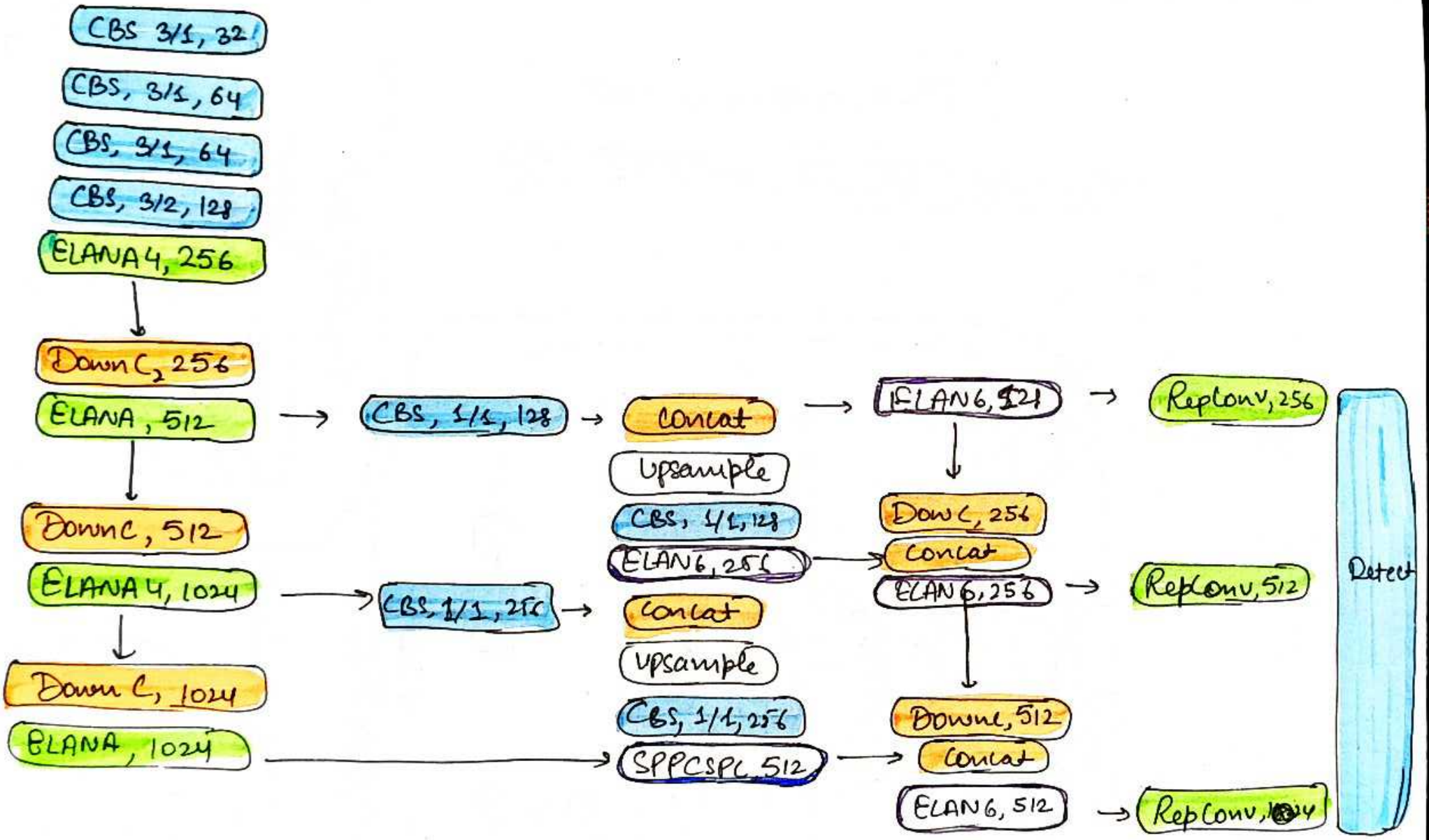








$$SiLU = x * \sigma(x) = x * \frac{1}{1 + e^{-x}}$$


CBS 3/1, 32
CBS, 3/1, 64
CBS, 3/1, 64
CBS, 3/2, 128
ELANA 4, 256
DownC, 256
ELANA, 512
DownC, 512
ELANA 4, 1024
Down C, 1024
ELANA, 1024
CBS, 1/1, 128
CBS, 1/1, 256
CBS, 1/1, 128
Concat
Upsample
CBS, 1/1, 128
ELAN 6, 256
Concat
Upsample
CBS, 1/1, 256
SPPCSPC 512
ELAN 6, 128
RepConv, 256
DownC, 256
Concat
ELAN 6, 256
RepConv, 512
DownC, 512
Concat
ELAN 6, 512
RepConv, 1024
Detect

Roboflow

Roboflow basically is an online tool, using which we can ANNOTATE the image or our data and download the data in the required format.

SIAMESE NEURAL NETWORK

Siamese Neural Network → Face Recognition
→ Signature Recognition .. etc

End Goal → Model (Image 1 , Image 2)
Changeable Image ← → Base Image

The output of the model

Image 1 = Image 2

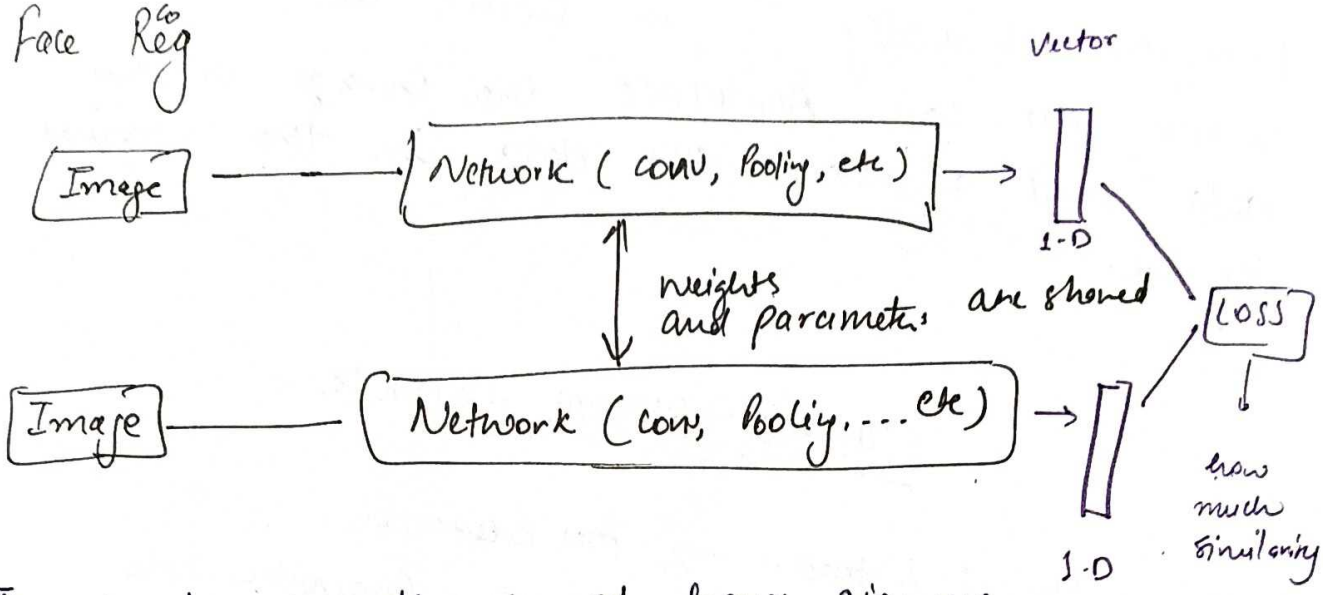
Image 1 ≠ Image 2

What we have to study?

→ Use case of
→ SIAMESE Neural Network
→ Architecture
→ Loss function → (i) Triplet loss func^n
(ii) contrastive loss

Siamese Neural Network

- Face Reg



The step-by-step approach of how siamese network architecture works we have two images and we want to compare and see if they are similar or dissimilar pairs.

1. The first subnetwork takes an image (A) as input and ~~we want to compare and etc if they are similar or dissimilar pairs~~ passes through convolutional layers and fully connected layers, we get a vector representation of the image.

2. Again pass the second image (B) through a network that is exactly the same with the same weights and parameters.

3. Now we have two encodings E(A) and E(B) from the respective images, we can compare these two to know how similar the two images are. If the images are similar then the encodings will also be quite similar.

4. we will measure the distance betⁿ these two vectors and if the distance betⁿ these is small then the vectors are similar or of the same classes and if the distance betⁿ is larger then the vectors are different from one another, based on the score.

Siamese

↳ we do not need lot of training data

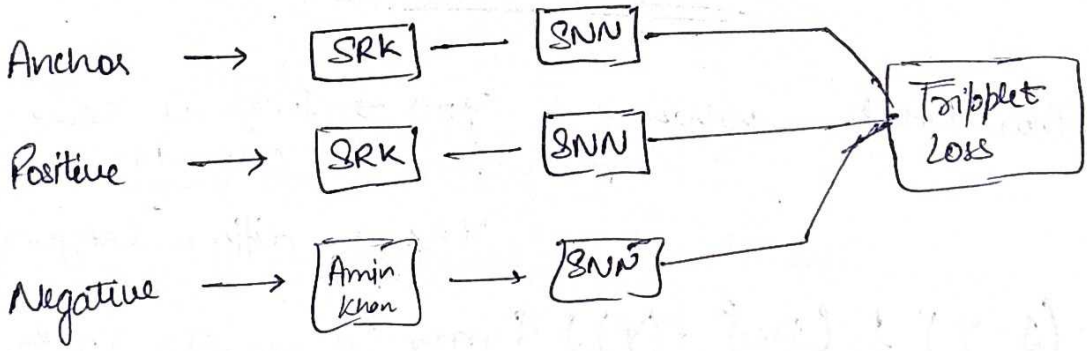
↳ Time ↑↑

There are 2 losses

1. Triplet Loss : Triplet loss is a loss function where a baseline is compared to a positive & a-ve.

$$\sum L(A,P,N) = \text{Max}\left(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha\right)$$

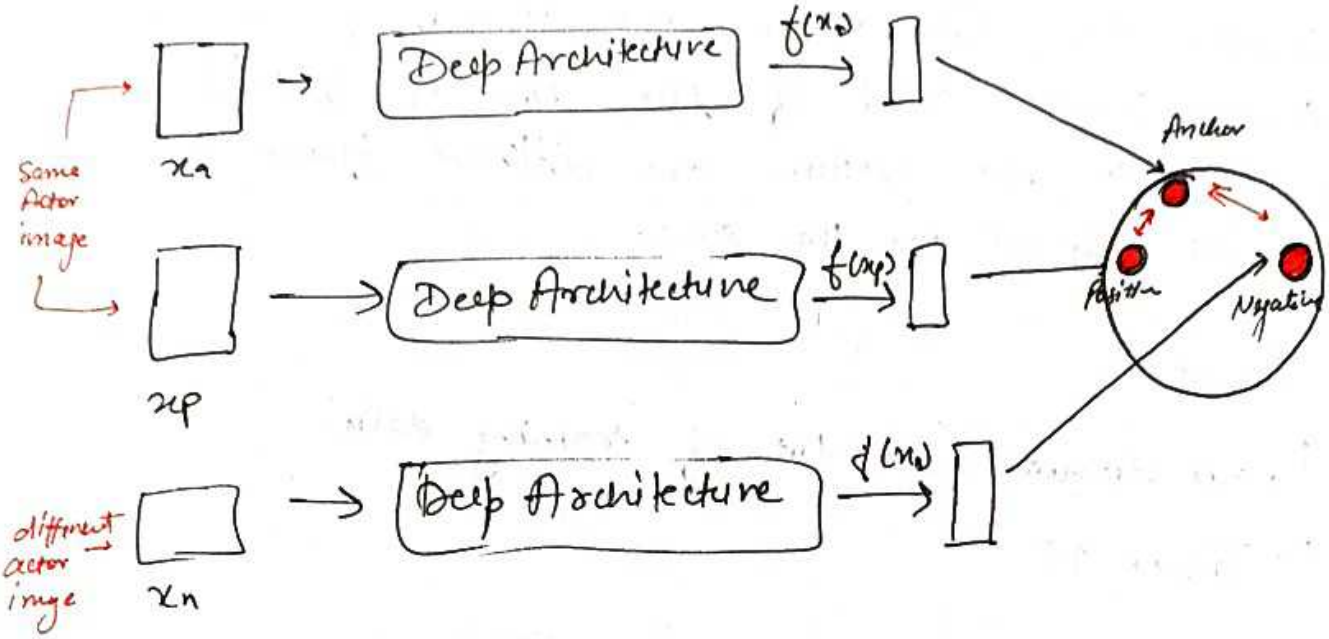
- max → (A, N)
- min → (A, P)

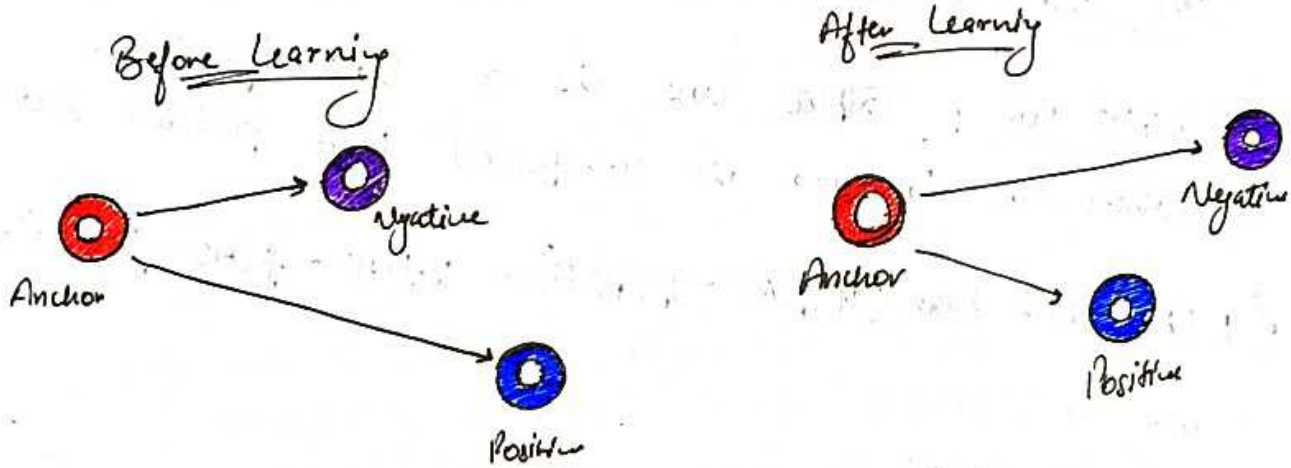


→ distance betⁿ Anchor and Positive is less.

→ distance betⁿ Anchor and Negative is high.

Triplet Loss with Keras and Tensorflow





Contrastive Loss

$Y \rightarrow$ True Label where $Y=0 \rightarrow$ Image of same categories

$Y=1 \rightarrow$ different categories

Eq^n - $(1-Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{\max(0, \underbrace{m}_{\text{margin (constant)}} - D_W)\}^2$

Case 1 → $Y = 0$

So, our eqn is

$$(1-Y)\frac{1}{2}(Dw)^2 + (Y)\frac{1}{2}\{\max(0, m-Dw)^2$$

for case 1 ←

Case 2 → $Y = 1$. different

→ Where Dw → Euclidean distance bet^n 2 Images

* In Triplet Loss we use Anchor, Positive and Negative

but in contrastive loss we use → Anchor, Positive

for similar Image

distance bet^n A, P is ↓↓

Anchor / Negative ←

for different Image

distance bet^n A and N ↑↑

Object Tracking

Video – 150 FPS

Real Time Object Tracking → Video

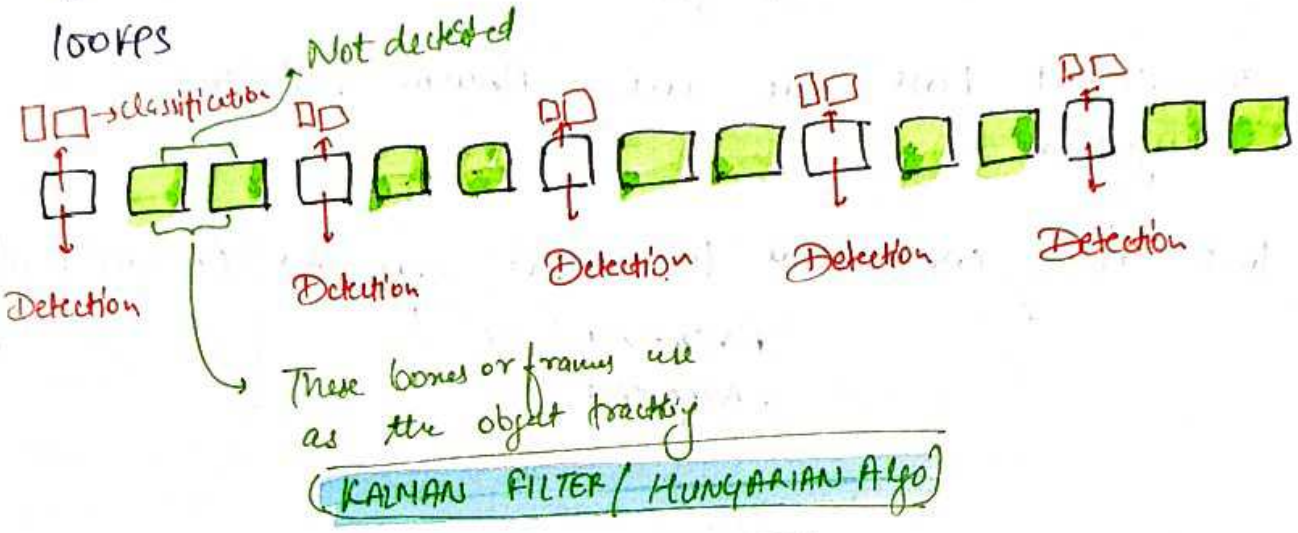
DeepSort → Object Tracking Algorithm

Object Tracking

Backbone → YOLO - V5 → 33.3 FPS

Video → helmets

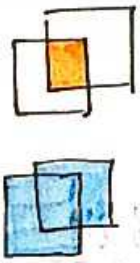
100 FPS



These boxes or frames will as the object tracking

(KALMAN FILTER / HUNGARIAN Algo)

$$IOU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



Kalman filter → Estimation

YOLO V5 → Detection

Object Tracking → IOU (Estimation & Detection)

Object Tracking → Similarity (Estimation & Detection)

(Siamese Neural Network)

Code Explanation

```
deepsort = DeepSort(cfg.DEEPSORT.REID_CKPT,
    max_dist = cfg.DEEPSORT.MAX_DIST, min_confidence = cfg.
    DEEPSORT.NMS_MAX_OVERLAP, max_iou_distance = cfg.DEEPSORT.
    MAX_IOU_DISTANCE,
    max_age = cfg.DEEPSORT.MAX_AGE, n_init = cfg.DEEPSORT.N_INIT
    , nn_buget = cfg.DEEPSORT.NN_BUDGET,
    use_cuda = True)
```

(capital letter)

cfg.DEEPSORT.REID_CKPT : This is the path to the checkpoint file of a deep neural network model used for ReID (Re-Identification). ReID is used to generate feature embeddings that capture the unique characteristics of objects, ending the algorithm to distinguish betn different objects based on their appearance

max_dist : Maximum Squared Mahalanobis distance betn the feature embeddings of two detection in order to consider them as the same object in the tracking process.

min_confidence : Minimum confidence threshold for a detection to be considered valid and used in the tracking process.

nms_max_ovilap : Non-Maximum Suppression (NMS) threshold to remove overlapping bounding boxes from the detection outputs before tracking.

max_iou_distance: Maximum Intersection over Union (IOU) distance both two boundig boxes to be considered for association in the tracking proces.

max_age: Maximum number of frames a track needs to be detection is keep alive without as assigned detection.

n_init: Number of frames a track needs to be detected in order to be confirmed and included in the tracking proces.

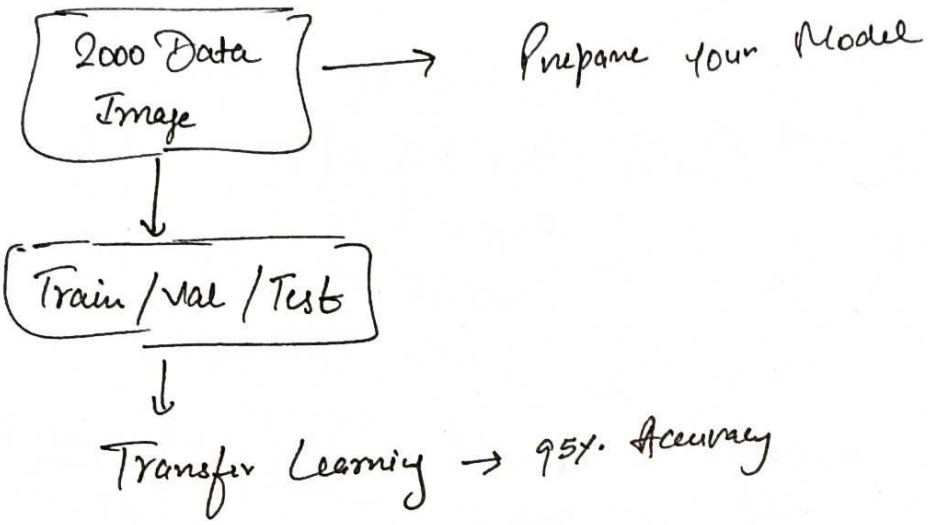
nn_budget: The maximum number of nearest neighbors that are considered when associating detection with tracks.

use_cude: A boolean flag indicating whether to use CUDA (GPU) acceleration if available.

In summary, this line of code initializes an instances of the DEEPSORT tracking algo with various configuration parameters. These parameters determine how detection are association with existing tracks, how track updates are performed, and how tracks, how track updates are performed and ReID embeddings are used for object tracking. The initialized deepsort object can then be used to perform object tracking on video frames using its tracking and updating methods.

Image Classification

Architecture



Functional Models

Problem with Sequential Model

ANN ↔ CNN

↳ Sequential

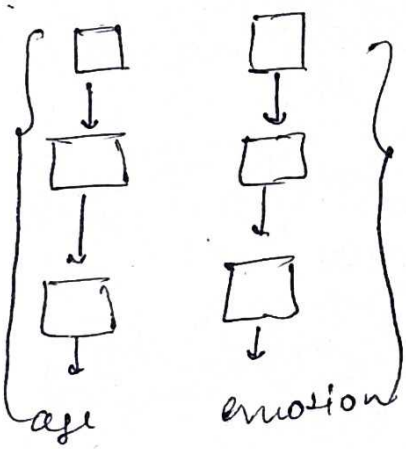
- 1 Input
- 1 output
- linear

linear topology

Example →  → Image dataset → human faces → 25000 images

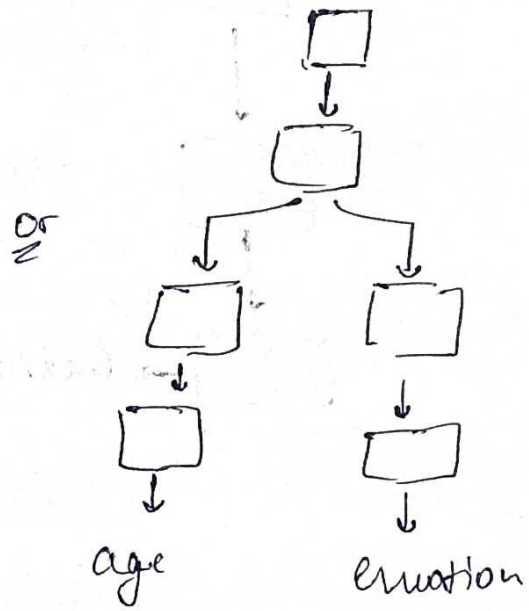
- age ?
- emotion ?
  - Sad
  - happy
  - angry

So, In this type of problem cannot solve sequential model. or we have to make 2 different model



age emotion

or



age emotion

Here's sequential model fail.

Example → E commerce website

Product Meta data (Numerical) (- - - - -)

Product description (textual)

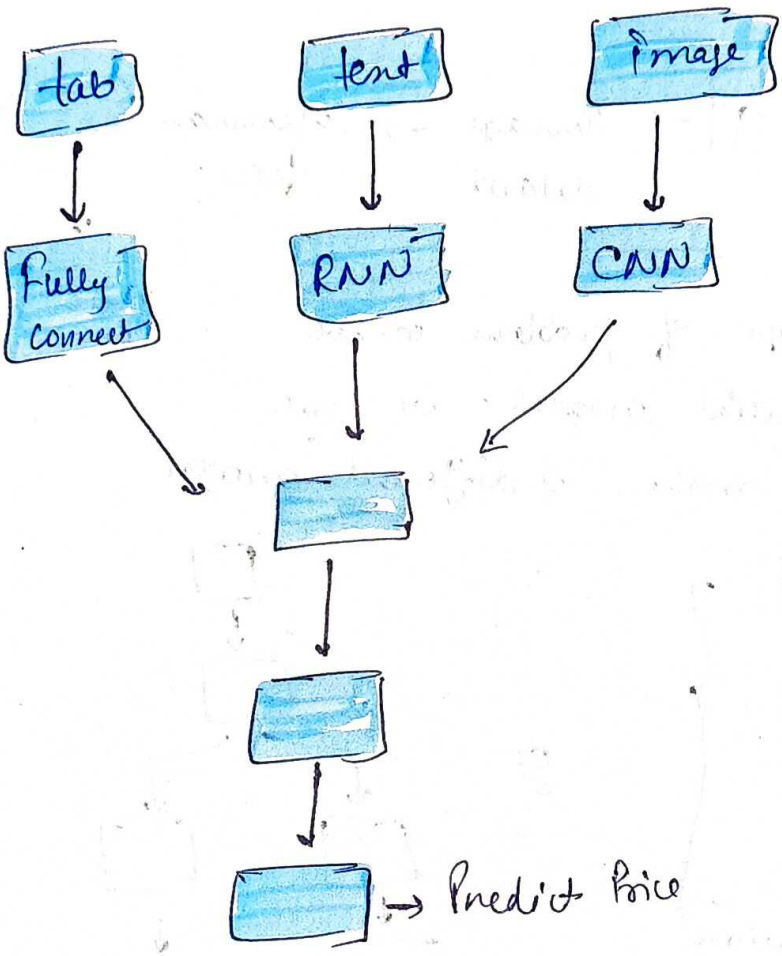
Product image

Predict Price

↑ tabular

↑ textual

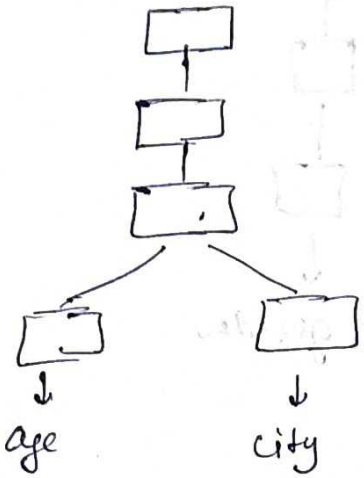
↑ Image



Let's apply Functional API on some data.

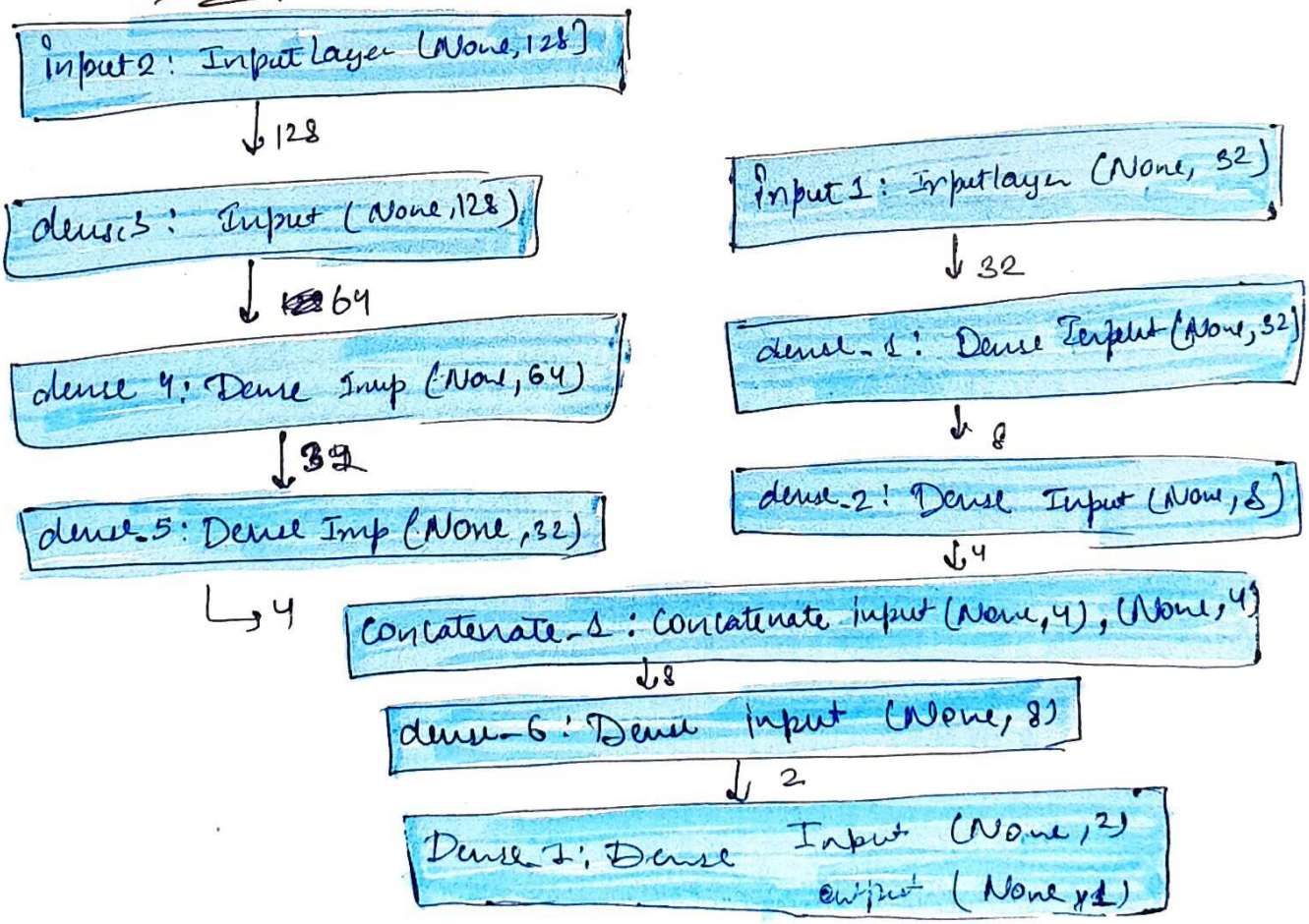
A Simple Example

Yearly Salary | weight | marital status | Predict → age, city (Delhi/Mumbai)



→ keras → Functional API

Multi Input Model



Multi Output Model

age / emotion → [A] → 20000 Images → gender
→ age

Transfer Learning
↓
VGG16

code