S13

Due No Due Date **Points** None **Available** after Apr 19 at 9:30am

YOLO 2 & 3

YOLOV2

Let's discuss some of the improvements w.r.t YOLOv1 in Yolo2

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

- YOLOv2 fixed v1's recall and localization problems

- By adding BN to **all the layers** it gets 2% improvement in mAP
- YOLOv2 is first fine-tuned on higher resolution images. This gives the network time to adjust its filters to work better on a higher resolution network. Then it is fine-tuned on detection. This gives 4% mAP improvement.
- YOLOv2 fully removes FC layers and use **anchor boxes** instead to predict bounding boxes.
- YOLOv2 predicts `class` as well as `objectness` for every anchor box.
- Anchor dimensions are picked using k-means clustering on the dimensions of original bounding boxes. Final anchor boxes are: (0.57273, 0.677385), (1.87446, 2.06253), (3.33843, 5.47434), (7.88282, 3.52778), (9.77052, 9.16828)
- The network predicts 5 bounding boxes and 5 coordinates for each bounding box: (t_x, t_y, $t_{w,}$ t_h and t_o)
- If the cell is offset from the top left corner of the image by c_x, c_y and the bounding box ground-truth/prior has width and height g_w, g_h then the predictions correspond to:
  - $bx = \sigma(t_x) + c_x$, where σ is sigmoid
  - $by = \sigma(t_y) + c_y$
  - $bw = g_w e^{tw}$
  - $bh = g_h e^{th}$
- YOLO predicts detection on a 13x13 feature map. This may not be sufficient to detect smaller objects. To fix this, they simply add a pass-through layer that brings features from an earlier layer at 26x26 resolution. The pass-through layer concatenates the higher resolution features with the low-resolution features by stacking adjacent features into different channels.
- Every 10 batches, the network chooses a random new image dimension size (multiples of 32) from 320x320 to 608x608.
- The final model, called Darknet-19 has 19 convolution layers and 5 max-pooling layers. 1x1 convolutions are used to compress the feature representations between 3x3.
- The network is first trained on classification for 160 epochs.
- After classification training, the last convolution layer is removed, and three 3x3 convolution layers with 1024 filters each followed by the final 1x1 convolution layer are added. The network is again trained for 160 epochs.
- During training both, detection and classification datasets are mixed. When the network sees an image with detection label, full back-propagation is performed, else only the classification part is back-propagated.

Where did our (TSAI's) concept of not connecting 2D layers with FC layers come from?

DARKNET-19

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

This table is designed to start at 224. If it starts at 448 (the actual training resolution), it would end at 13x13.

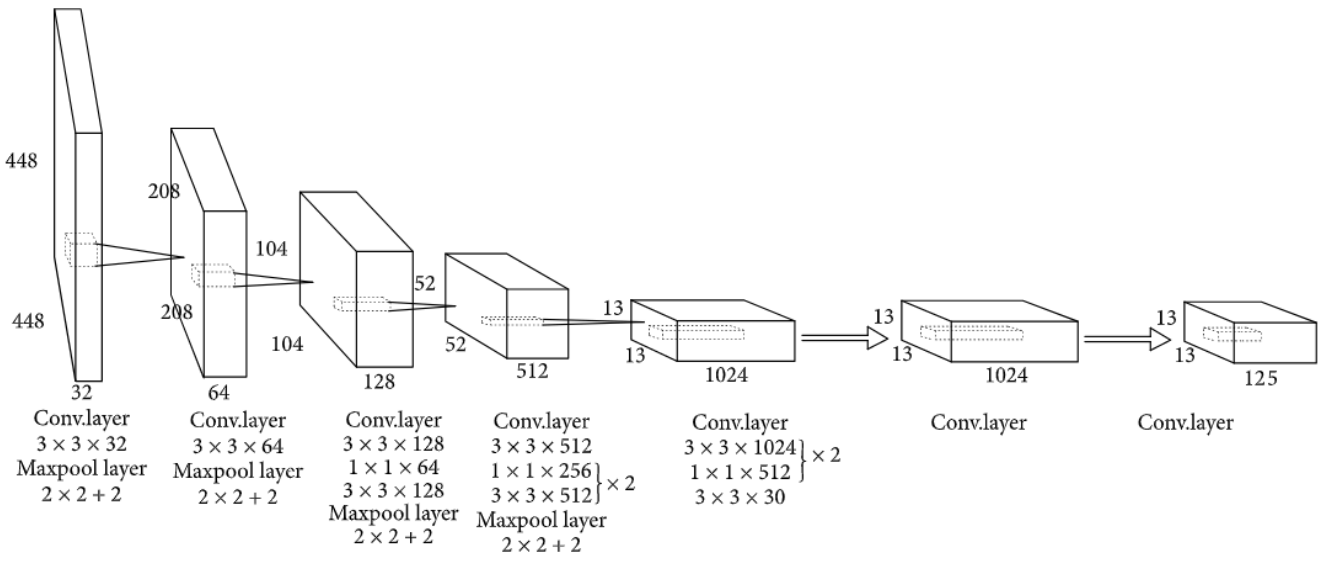
Now how do we get the 4D data as we see in this image below?

Let us observe this image again:

We have 5 anchor boxes, and each one predicts 25 numbers.

So we need 13x13x5x25

We can re-write this as **13x13x125**! (remember Pixel Shuffle Algorithm?)



YOLOV2 Loss Function Recap

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

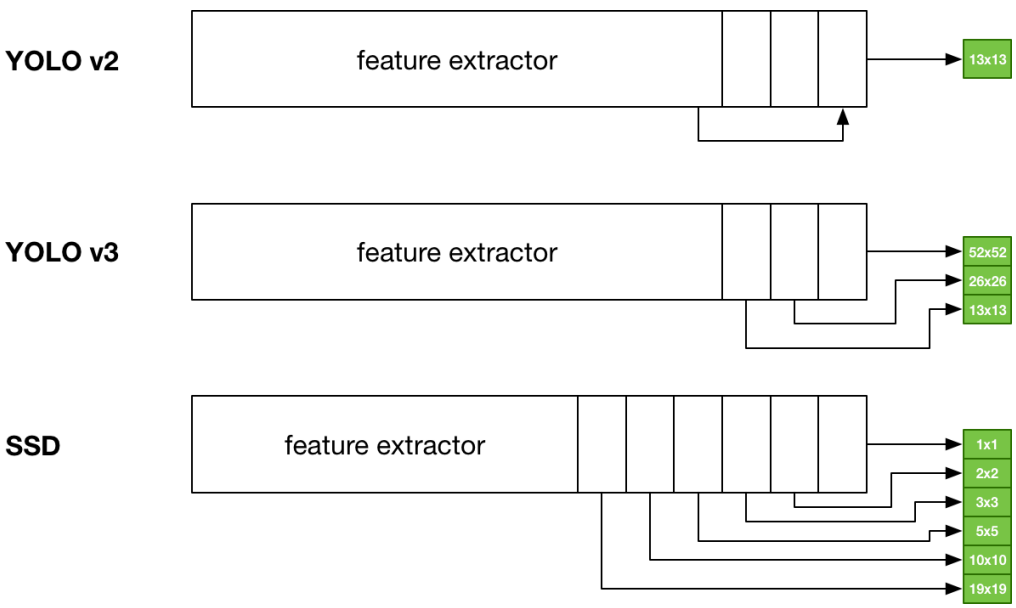
$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

YOLOV3



Complete Architecture

V3 uses only Convolutional layers, not even the pooling layer! How can we avoid any pooling layer?

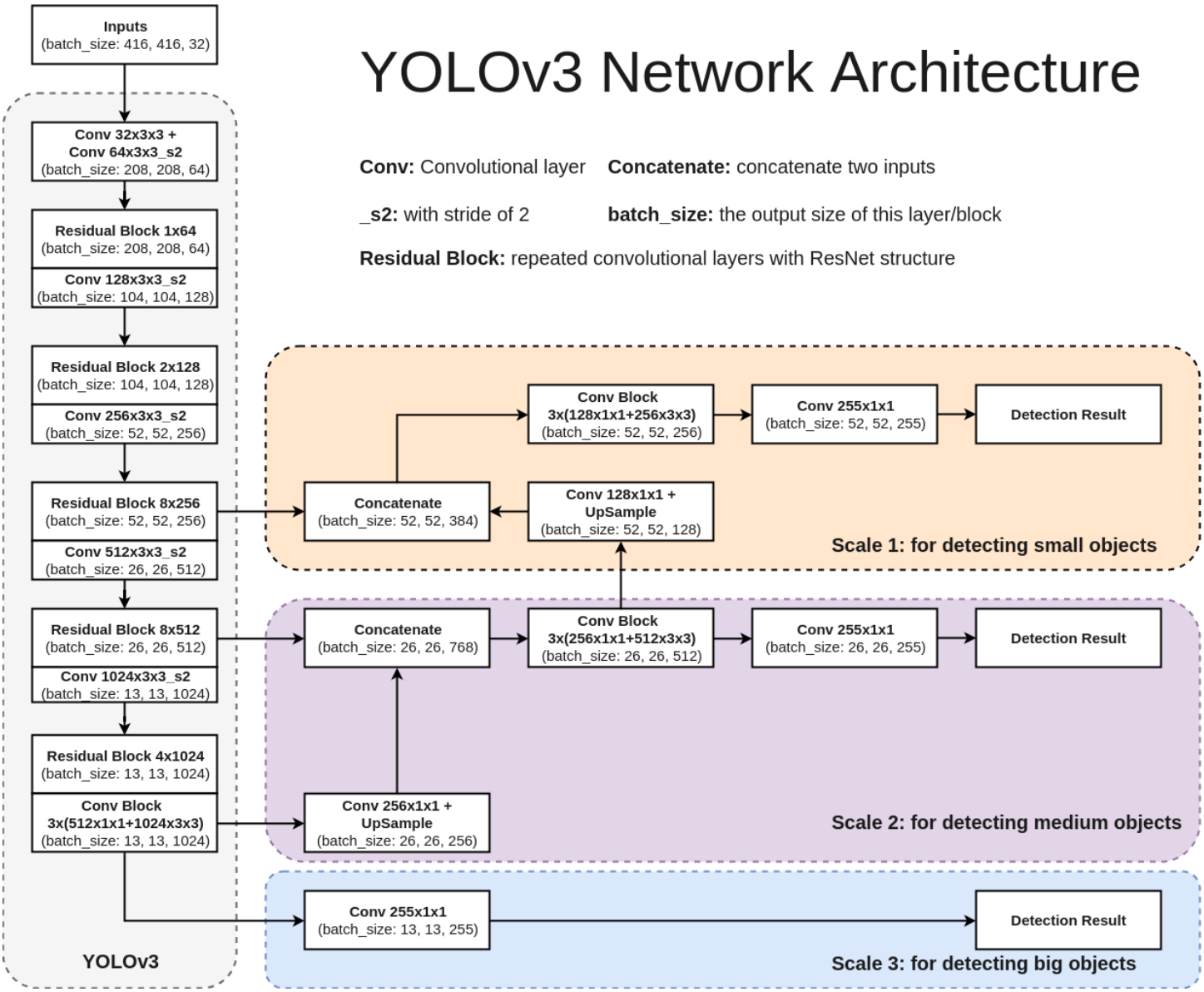
A 3x3 with a stride of 2.

If we start at 416 and end at 13, we have taken a total stride of 32 (416/13).

COCO Dataset has 80 classes. So final output shall be

13x13x**3**x(4+1+80) = 13x13x255

YoloV3 has 3 Anchor Boxes! or 9?



Upsampling Layer

```
void upsample_cpu(float *in, int w, int h, int c, int batch, int stride, int forward, float scale, float *out)
{
    int i, j, k, b;
    for(b = 0; b < batch; ++b){
        for(k = 0; k < c; ++k){
            for(j = 0; j < h*stride; ++j){
                for(i = 0; i < w*stride; ++i){
                    int in_index = b*w*h*c + k*w*h + (j/stride)*w + i/stride;
                    int out_index = b*w*h*c*stride*stride + k*w*h*stride*stride + j*w*stride + i;
                    if(forward) out[out_index] = scale*in[in_index];
                    else in[in_index] += scale*out[out_index];
                }
            }
        }
    }
}
```

Looks complicated?

That's just a simple zoom. 1 pixel is transformed into 2x2 pixel :|

Class Confidence

YoloV3 has an interesting take on Class probabilities.

Normally you'd take a SoftMax of the output vector. This is based on the assumption that classes are mutually exclusive.

If it is a **Dog**, *it cannot be a* **Cat**!

YOLOv3 asks a question, what if we have classes which are not mutually exclusive

If it is a ***Person***, it may be a ***Man*** as well!

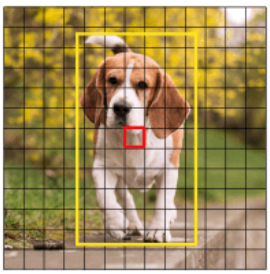
So instead of SoftMax, v3 uses a sigmoid function.

Predictions over 3 scales

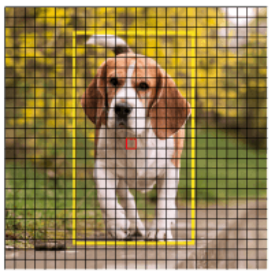
v3 makes predictions at 52x52, 26x26 and 13x13; alternatively,

at **strides of 8, 16 and 32.**

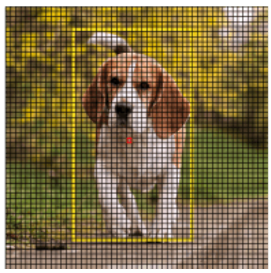
Prediction Feature Maps at different Scales



13 x 13



26 x 26



52 x 52

The network downsamples the input image until the first detection layer,

where are **detection is made** using feature maps of a layer with stride 32.

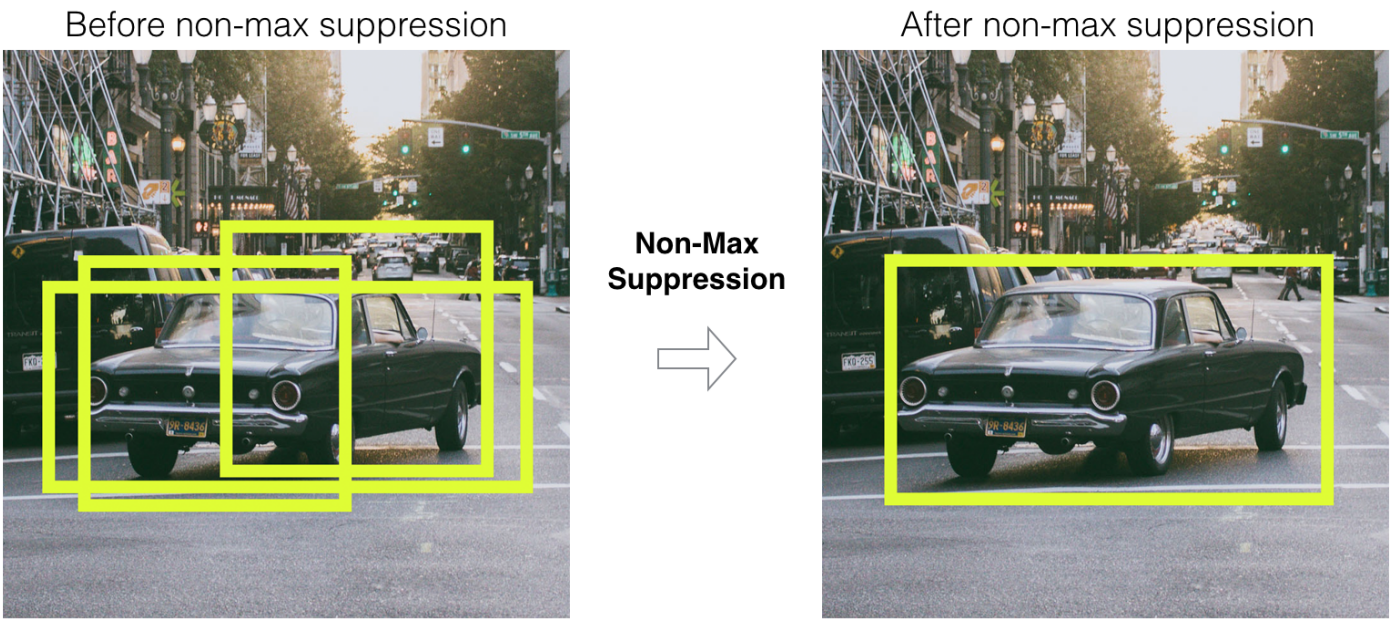
Then, layers are upsampled by a factor of 2 and concatenated with feature maps of previous layers with identical feature map size. Another **detection is made** at stride of 16.

Then, the same upsampling procedure is repeated and a final **detection is made** at the layer of stride of 8.

Output Processing

v3 in total now predicts (52x52 + 26x26 + 13x13)*3 = **10647** bounding boxes

Object Confidence Threshold: We filter the boxes based on their objectness score.



Non-maximum Suppression: A technique that helps selects the best bunding box among overlapping proposals.

Assignment Background:

We are going to implement full YoloV3 training, but that's tough and involves too many steps. So instead of giving it as one single assignment, we are going to build up to it slowly. We'll first look at YOLOV2 Orignal steps taken by Authors, and then move to YoloV3 on Pytorch training.

Assignment:

1. OpenCV Yolo: **SOURCE** (https://pysource.com/2019/06/27/yolo-object-detection-using-opencv-with-python/)
    1. Run this above code on your laptop or Colab.
    2. Take an image of yourself, holding another object which is there in COCO data set (search for COCO classes to learn).
    3. Run this image through the code above.
    4. Upload the link to GitHub implementation of this
    5. Upload the annotated image by YOLO.
2. Training Custom Dataset on Colab for YoloV3
    1. Refer to this Colab File: **LINK** (https://colab.research.google.com/drive/1LbKkQf4hbluiUHunLlvY-cc0d_sNcAgS)
    2. Refer to this GitHub **Repo** (https://github.com/theschoolofai/YoloV3)
    3. Collect a dataset of 500 images and annotate them. **Please select a class for which you can find a YouTube video as well.** Steps are explained in the readme.md file on GitHub.
    4. Once done:

    1. **Download** (https://www.y2mate.com/en19) a very small (~10-30sec) video from youtube which shows your class.
    2. Use **ffmpeg** (https://en.wikibooks.org/wiki/FFMPEG_An_Intermediate_Guide/image_sequence) to extract frames from the video.
    3. Upload on your drive (alternatively you could be doing all of this on your drive to save upload time)
    4. Inter on these images using detect.py file. **Modify** detect.py file if your file names do not match the ones mentioned on GitHub.
       `python detect.py --conf-thres 0.3 --output output_folder_name`
    5. Use **ffmpeg** (https://en.wikibooks.org/wiki/FFMPEG_An_Intermediate_Guide/image_sequence) to convert the files in your output folder to video
    6. Upload the video to YouTube.
5. Share the link to your GitHub project with the steps as mentioned above
6. Share the link of your YouTube video
7. Share the link of your YouTube video on LinkedIn, Instagram, etc! You have no idea how much you'd love people complimenting you!

Questions on the submission page asked are:

1. Upload the link to your YOLOv3OpenCV code on Github. - 100 pts
2. Upload the link to the image annotated by OpenCV YOLO inference. - 100 pts
3. Share the link to your GitHub project with the steps as mentioned above (for YoloV3 training on Colab). -1000 pts
4. Share the link of your YouTube video (your object annotated by your YoloV3 trained model). - 800 pts.

Session Video: