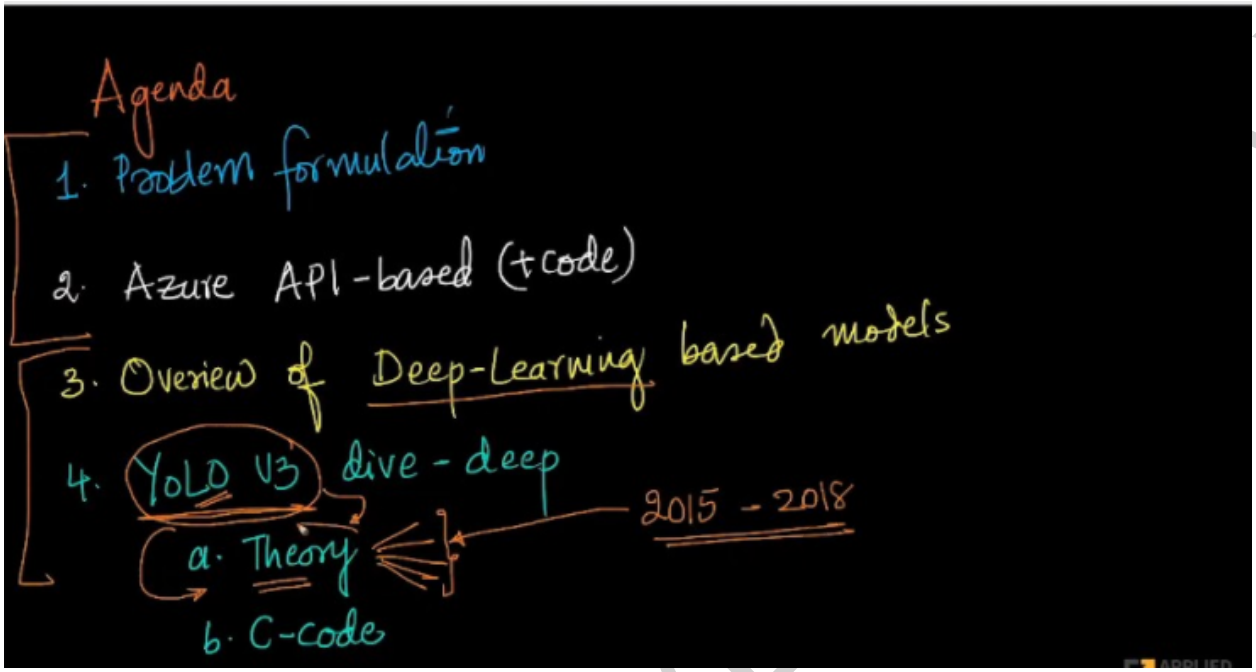
69.1 Object Detection

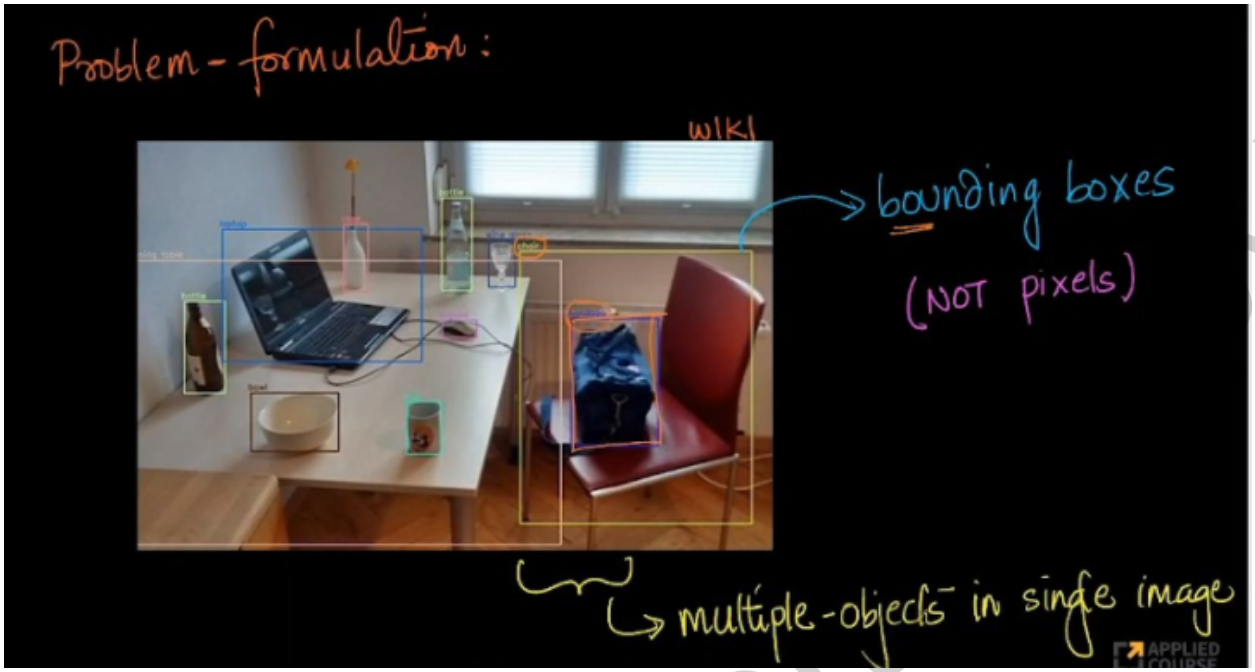


Timestamp : 03:37

Agenda :

1.) **Problem formulation** : We will first understand the problem and it's trade-offs.
2.) **Azure API-based** : We will understand the same by implementing in Azure. Also, other things like performance-metrics are discussed.
3.) **Overview of Deep-learning based models** : We will understand how to implement object detection using neural networks.
4.) **Yolo V3 deep dive** : We will discuss the theory and implementation of Yolo-V3. Also, we will understand the implementation of Yolo-V3 in C language.

Problem formulation :

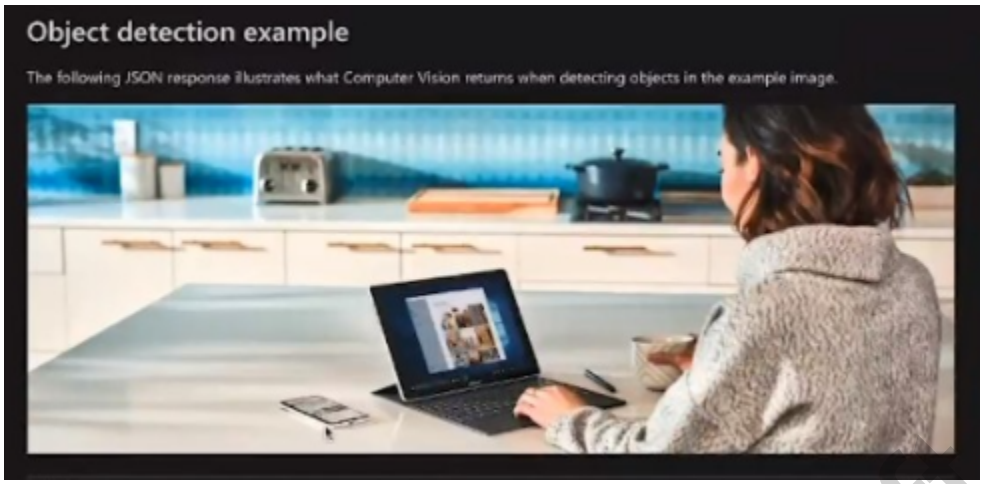


Timestamp : 06:08

Given an image, we need to draw bounding boxes over all the possible objects in it. For example, in the above image, there are multiple objects like laptop,chair,cup ,etc. For each of those objects, a bounding box is drawn. A bounding box is nothing but a rectangle which encloses the object in an image. It's quite similar to image segmentation, but not the same. In image segmentation, rather than drawing or finding the bounding boxes, we find all the pixels which are related to an object. For example, rather than finding a bounding box for the laptop, we will find all the pixel values which belong to the laptop. Segmentation algorithms are more time consuming than object detection algorithms since the former works on pixel level.

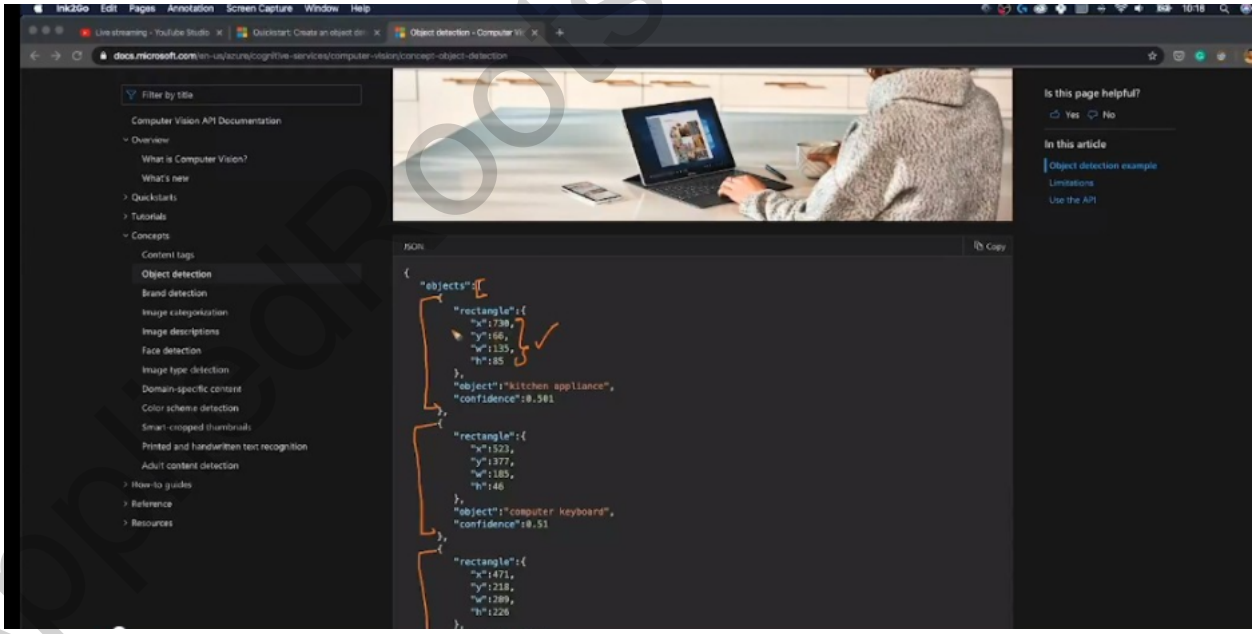
So, for an object detection algorithm, the input is the image and the output is the set of bounding boxes and the corresponding objects which the bounding boxes enclose.

Azure API : The prerequisite for understanding this is knowledge of using web api in python. Azure API is easier to test various algorithms and the process is much more simpler.



Timestamp : 11:01

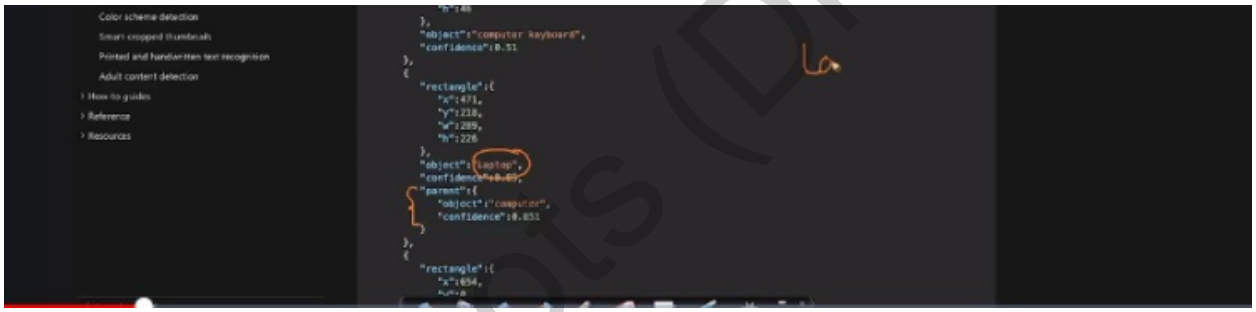
So, the Azure API takes in an image as the input. For example, consider the above image as the input. Now, the API needs to return the bounding boxes which cover all possible objects in the image. This API returns the bounding boxes in JSON format.



Timestamp : 12:18

In the above image, we can see a json dictionary has been returned by the API. The first key is “objects”. This key contains a list of objects. Each object contains the information about a bounding box.
“rectangle” : It denotes that the shape of the bounding box is a rectangle.
“x” : The centre x coordinate of the bounding box.
“y” : The centre y coordinate of the bounding box.
“w” : Width of the bounding box.
“h”: Height of the bounding box.
“object” : Name of the object contained in the bounding box.
“confidence” : Confidence level of the object being present in the bounding box.

It also predicts the parent of an object.



Timestamp : 14:33

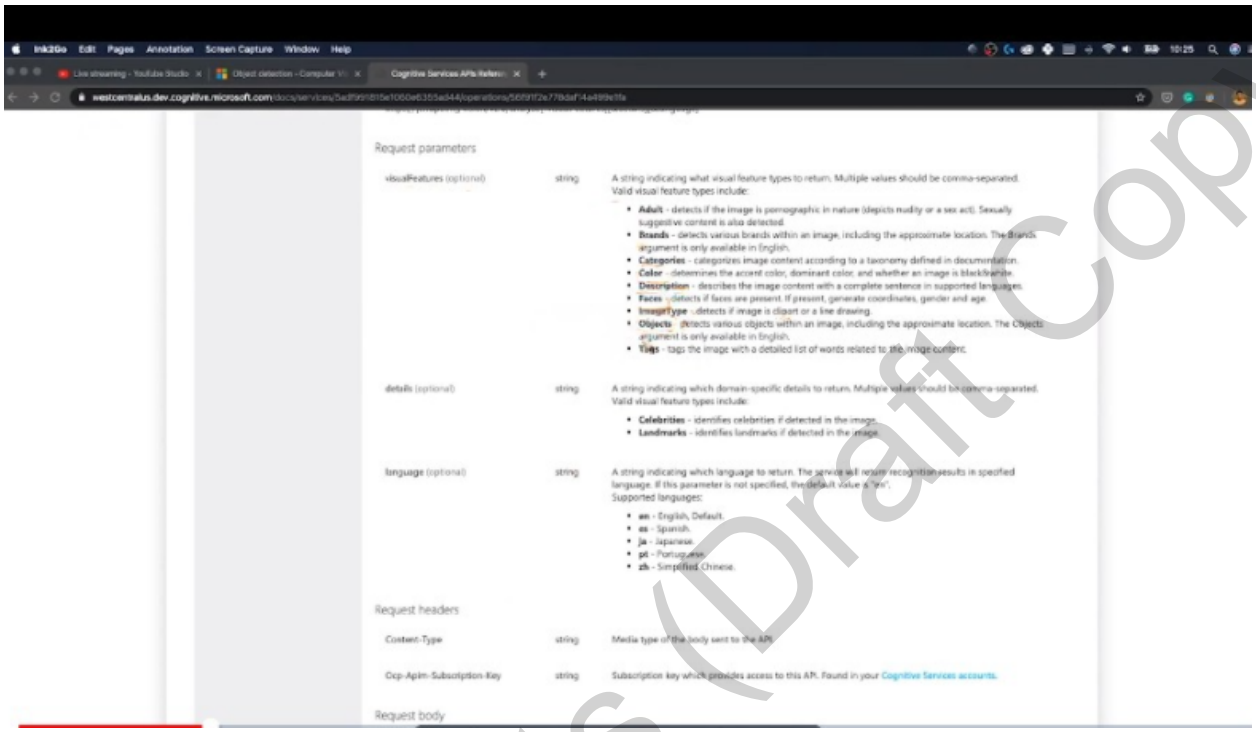
From the above image, we can see that the parent for the object laptop is a computer. So, it also predicts a laptop is a computer and not a single object by itself. To be precise, it unwraps the hierarchy of an object.

There are some limitations using Azure API. Some of them are listed below.

1. Objects are generally not detected if they’re small(less than 5% of the image).
2. Objects are generally not detected if they’re arranged closely together. Because, the bounding boxes may overlap and it’s hard for the model.

3.) Objects are not differentiated by brand or product names.

We will first understand the Request parameters.



Timestamp : 19:14

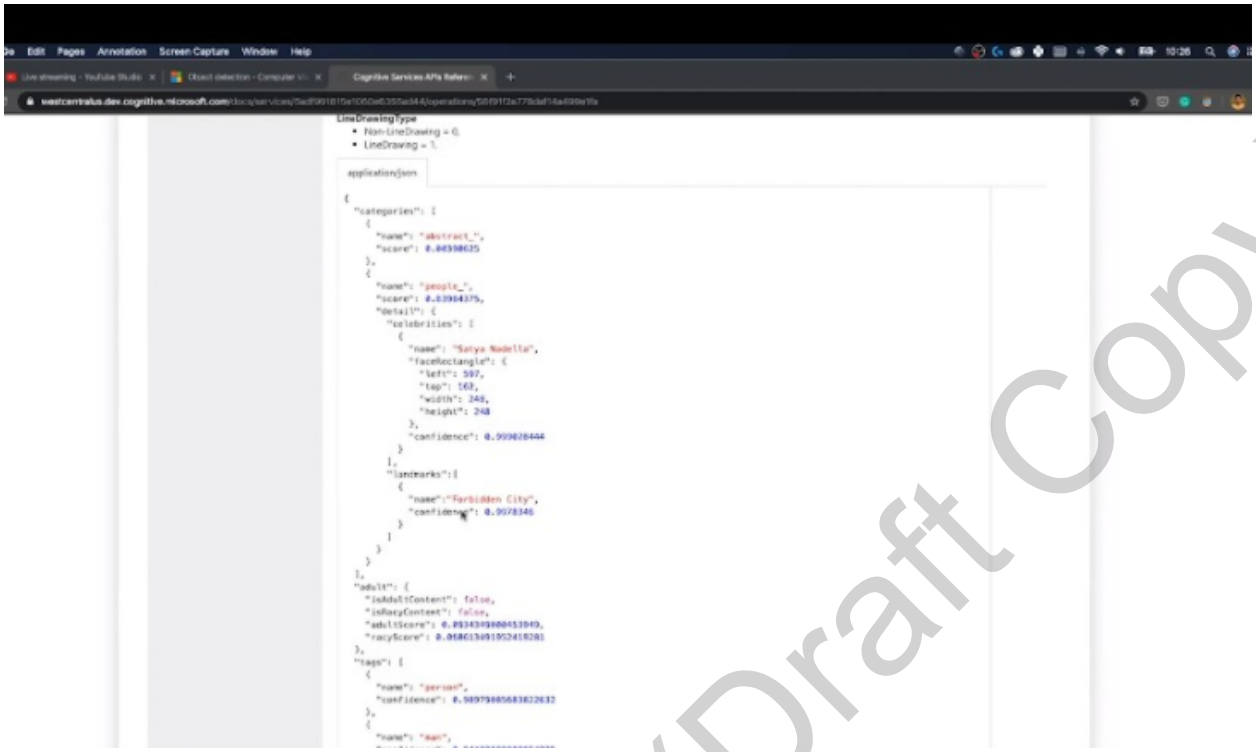
In the above image, we can see one of the parameters as visualFeatures.

This takes in a string where we need to specify what we want.

In this session, we will need Objects. So, we pass “Objects” in the visualFeatures parameter.

language : You can choose the language in which you want the output description to be.

A sample json returned by this API.

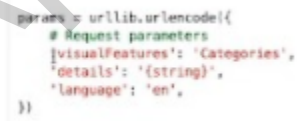


Timestamp : 20:28

As we already mentioned about json format, it's easier to interpret now. It contains the information about the bounding boxes. It's self-explanatory.

Code sample.

We will give the appropriate requests headers.



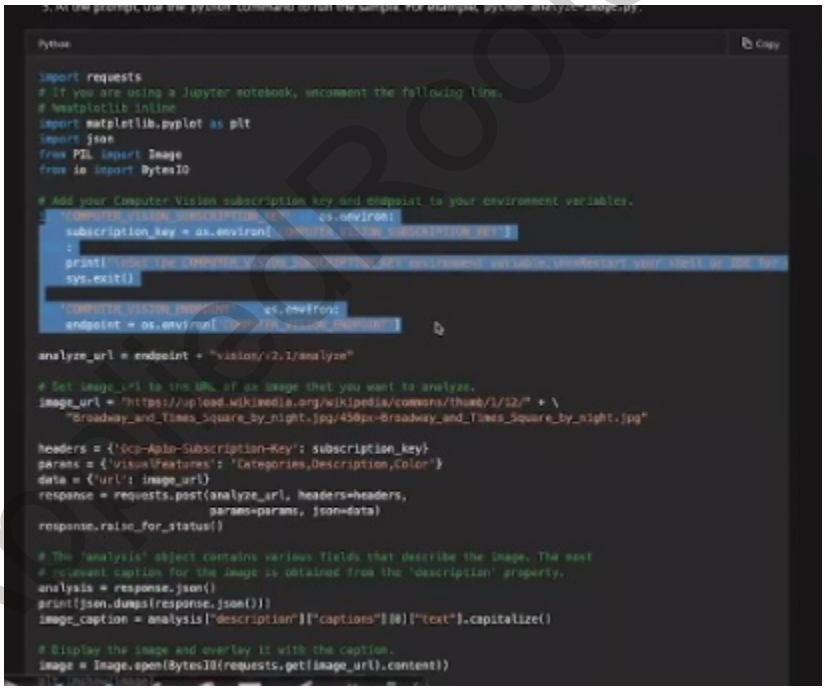
If we are going to perform object detection, then we need to pass “Object” in the visualFeatures parameter.

```
try:
    conn = http.client.HTTPSConnection('westcentralus.api.cognitive.microsoft.com')
    conn.request("POST", "/vision/v2.0/analyze?%s" % params, "{body}", headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
```

Then, we open a HTTPS connection and make a request on the azure API. The Azure API upon validating the request, returns a response. We already saw that the response is in JSON format. Then, we print the data to the console and close the connection.

If we want the output to be well formatted, then try using the below code snippet.

print(json.dumps(response.json()))



Timestamp : 25:27

As usual, we are importing a bunch of libraries.

Then, we define some environment variables. This has been covered in one of the previous live sessions. Note : Please check the prerequisite.

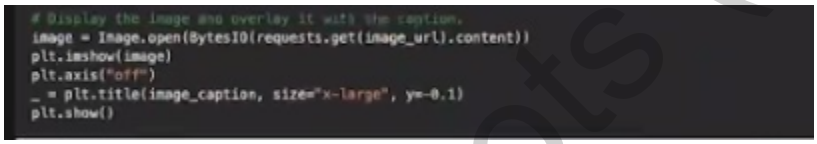
Then, we need to specify the url of the image.

In the params, we can pass “Object” to the key visualFeatures if we want to perform object detection.

Then, we simply post our request.

Then, we can jsonify the response using json().

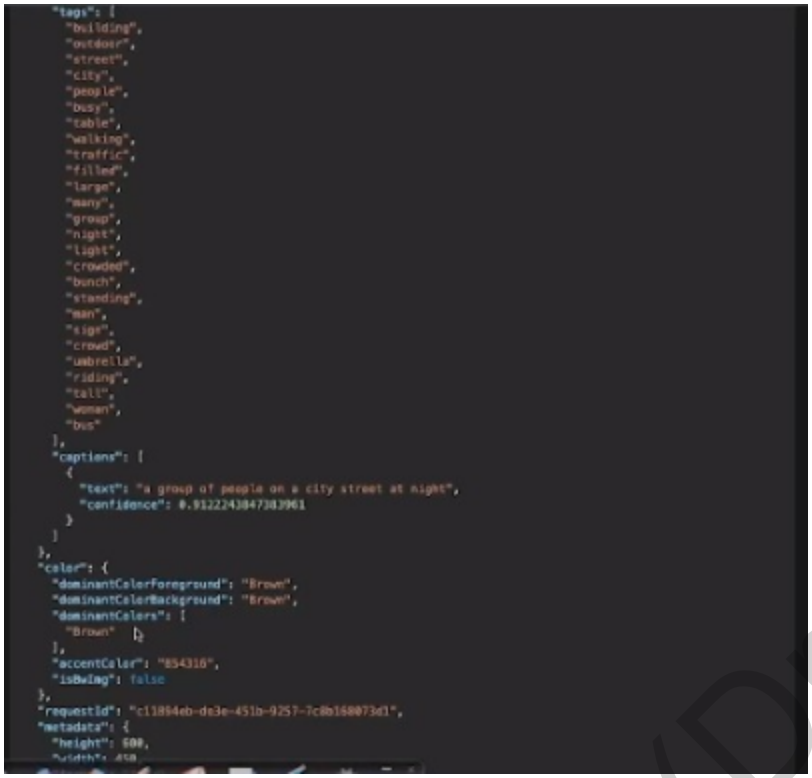
Once done, we can display the image with the caption being returned from the description parameter.



```
# Display the image and overlay it with the caption.
image = Image.open(BytesIO(requests.get(image_url).content))
plt.imshow(image)
plt.axis("off")
_ = plt.title(image_caption, size="x-large", y=-0.1)
plt.show()
```

We are displaying the image with the title being the caption contained in the description parameter.

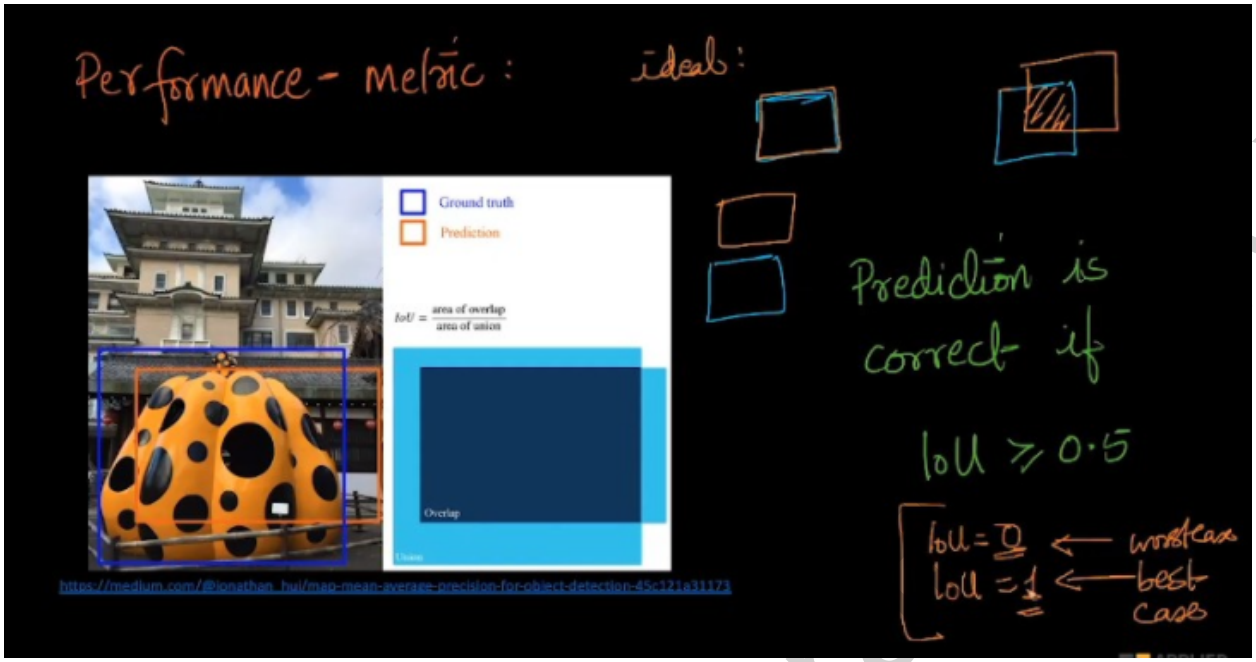
A sample response.



Timestamp: 26:30

There is no information about the objects in the returned response since we didn't specify "Objects" in the visualFeatures parameter.

Performance metric :



Timestamp : 32:27

One of the performance metrics for evaluating object detection algorithms is IOU (Intersection Over Union).

In the above image, the blue bounding box is the actual one or ground truth. It's what is present in our dataset. The bounding box colored with yellow or anchor box is what the model predicts. We want to devise a measure which compares the two bounding boxes. IOU is one such measure which compares the two bounding boxes.

IOU = area of overlap / area of union.

First, we find the area of intersection between the two bounding boxes. Let's call it I.

Then, we find the union of the area of the two bounding boxes. Let's call it U.

IOU is nothing but I/U.

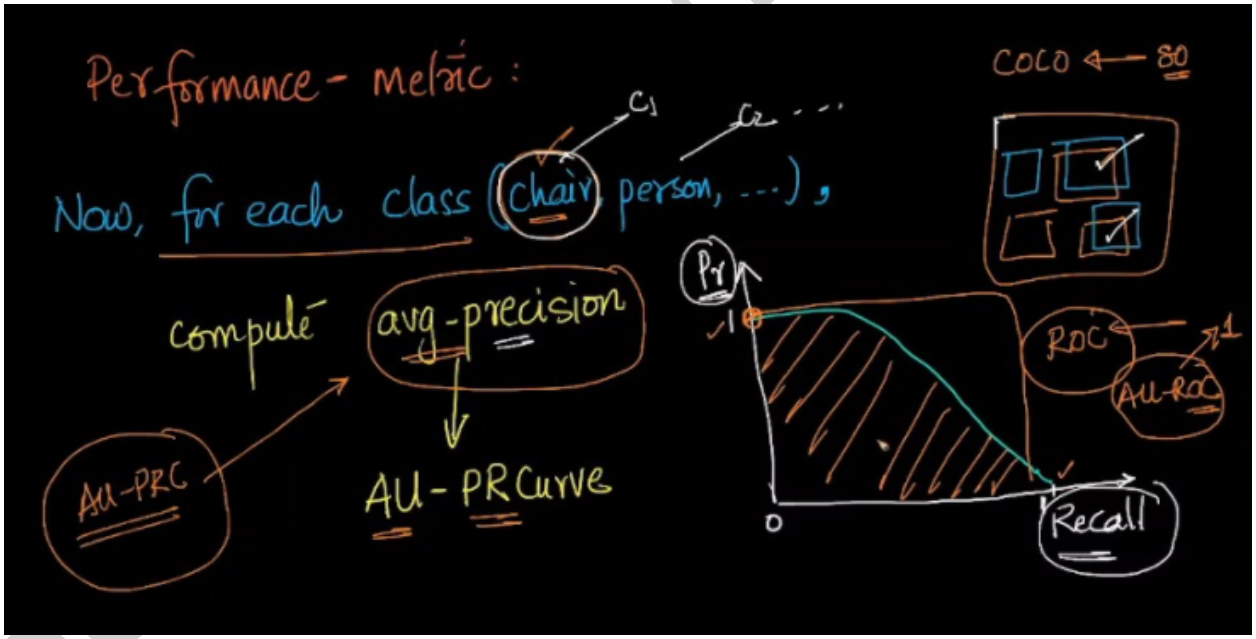
Is the IOU score bounded ?

Yes. The maximum value I can take is U. So, the maximum value would be 1. This means, both the ground truth bounding box and the predicted one are the same. This is the ideal case we want. The minimum value it can take is 0. This means, the value of I is 0 i.e, there is no intersection. Note : U can't take the value 0. This is the worst case we want to avoid.

If IOU>=0.5, then we can have a confidence level of 0.5 of being close to the actual bounding box. It's similar to the decision rule we make in logistic regression.

So, if IOU>=0.5, then we say the prediction is correct, else incorrect.

How to deal with multiple classes?



Timestamp : 40:17

Let's say we have n classes where each class is represented as ci where it ranges from 1 to n.

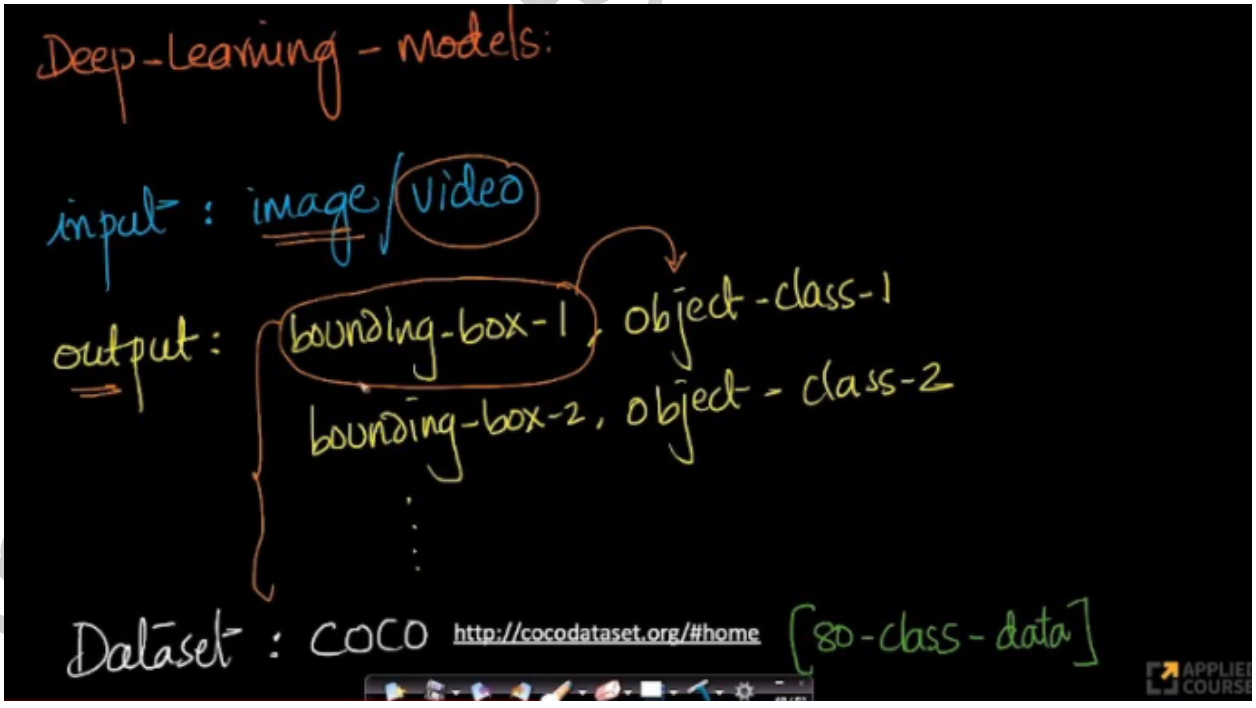
For each class, we will plot the precision vs recall curve with varying thresholds. The concept of precision and recall is already covered in our course. One property of precision vs recall curve is that, we can get a precision of 1 with recall of 0 and vice versa.

Then, we compute the area under the curve which is named as average precision.

So, for each class ci, we compute the above measure. Let's say for ci, the average precision is ai.

Mean Average Precision (mAP) : We simply compute the mean of ai's.

Implementing object detection through deep learning



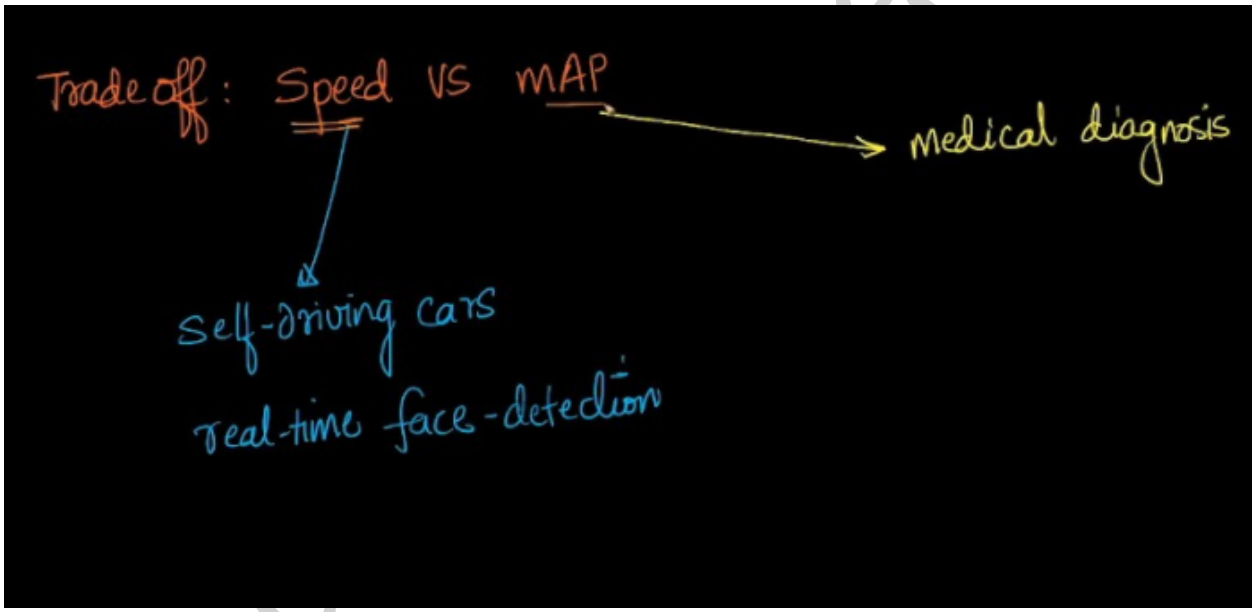
Timestamp : 54:18

Input : Image/Video. Typically it's an image. But, video is nothing but a sequence of frames. So, we can pass each of those frames to the model and get the predictions.

Output : (Bounding box - 1 , Object - class -1 , … Bounding box - i , Object - class - i) . Basically it returns a set of bounding boxes with the class of the object it encloses. We already discussed the json format.

Dataset : COCO (80 - class dataset). It's a fairly large dataset.

Next, we will discuss the trade-off.



Timestamp : 55:09

The trade-off is between speed and mAP.

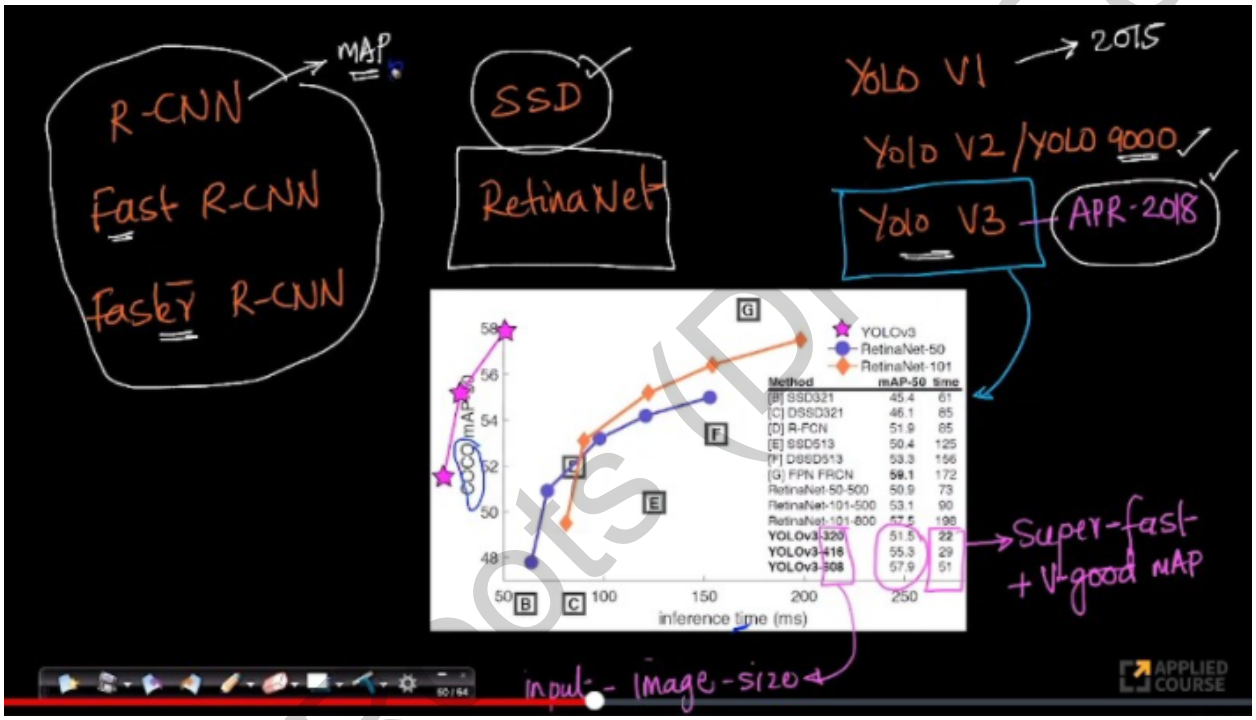
Speed : Given an input image, how fast the algorithm can return the information about the bounding boxes and the respective classes.

mAP : Mean average precision. We already discussed it.

For real-time face detection, speed really matters.

For applications involving medical diagnosis, mAP is more important than speed.

mAP matters where the cost for error is high.



Timestamp : 59:26

There are a bunch of algorithms which implement object detection.

Some of them are listed above in the image.

For various algorithms , we are plotting its mAP and inference time.

R-CNN is generally considered to be best for mAP.

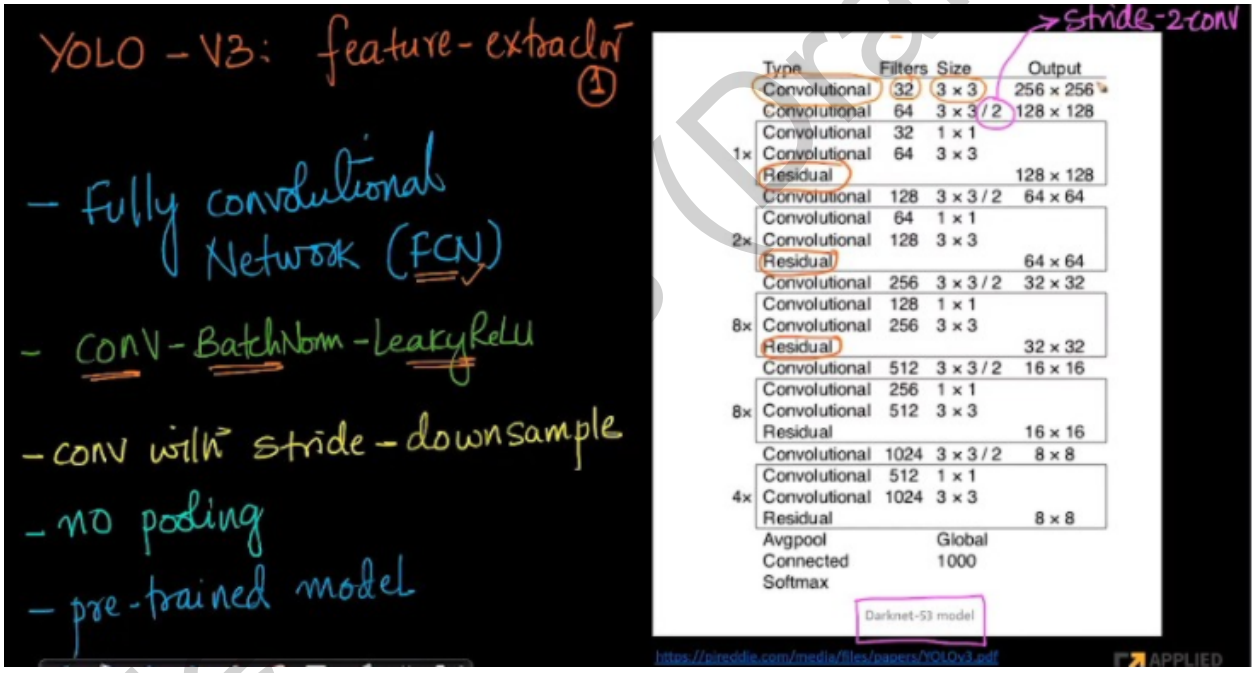
We already discussed the trade-off between mAP and inference time.

Ideally, we want a less inference time and higher mAP. But from the plot given above, we can see there is a clear trade-off between them.

Yolov3-320 seems to be super fast and holds a very good mAP. Here, 320 refers to the image size. Please draw conclusions from the table given above.

From the above table, we can see that YOLO is the best algorithm.

Let's dive into the architecture of YOLO-V3.



Timestamp : 01:07:20

YOLO-V3 has multiple components. The first component is the feature extractor. As the name suggests, it extracts a set of features from the given image. In the context of transfer learning, we learnt that after removing the final dense layer from the pretrained network, we can use the rest of the network as a feature extractor.

The model or architecture which is used in YOLO-V3 is Darknet-53. It has 53 layers of convolutions.

1.) It's a fully convolutional neural network. This means, there is no dense layer present in the network. This reduces the time complexity.
2.) Each layer does convolution followed by batch normalization and then applying an activation function which is LeakyReLU.

The size of the input image is 256. First, it goes through a convolution operation with 32 filters and a kernel with shape 3*3. The output is of size 256.

In the second stage, the number of filters used are 64 and with a 3*3 kernel. Here, we use stride as 2, so that the size of the output is half of the size of the input. So, here the output size is 128 which is 256/2. Basically in stride 2 convolution, we translate the convolutional kernel in the image 2 pixels ahead rather than 1 pixel at a time. It downsamples the image. Because, it will reduce the complexity. Also , downsampling ensures that only the required features are learnt since the size is getting reduced. So, the model is forced to extract only the required features.

There is no pooling layer used in this architecture.

We also use the residual connection here. We discussed this while learning about ResNet. Residual connections avoid overfitting in a deep neural network.

1X simply means, we are going to use this layer one time.

Generally , iX means we are going to repeatedly use this layer i times in series.

YOLO-V2 had Darknet-19 as it's backbone. But, it doesn't perform well as the research suggests.

In the final layer, we do global average pooling and produce a 1000 dimensional vector using a dense layer which is nothing but the logits and its passed to the softmax function. Then, they train this network on the ImageNet dataset. First, they are training the model so that it recognizes the object rather than localizing it in object detection.

Timestamp : 01:15:43

Bn Ops simply means how many billion operations we can perform in a second using this architecture.

BFLOP/s simply means how many billions of floating operations we can perform in a second using this architecture.

FPS simply means how many frames the model can process in a second.

We want to have best in both i.e, in speed and in accuracy.

We want to have less Bn Ops and high BFLOP/s. Also, we want high FPS.

Already ResNet based models were good in accuracy. But, they were slower. So, it can't be implemented in real-time. Most of the applications involve real-time. So, Darknet-53 was designed to overcome the above limitation. But YOLO-V3 is slower than YOLO-V2. There's always a trade-off.

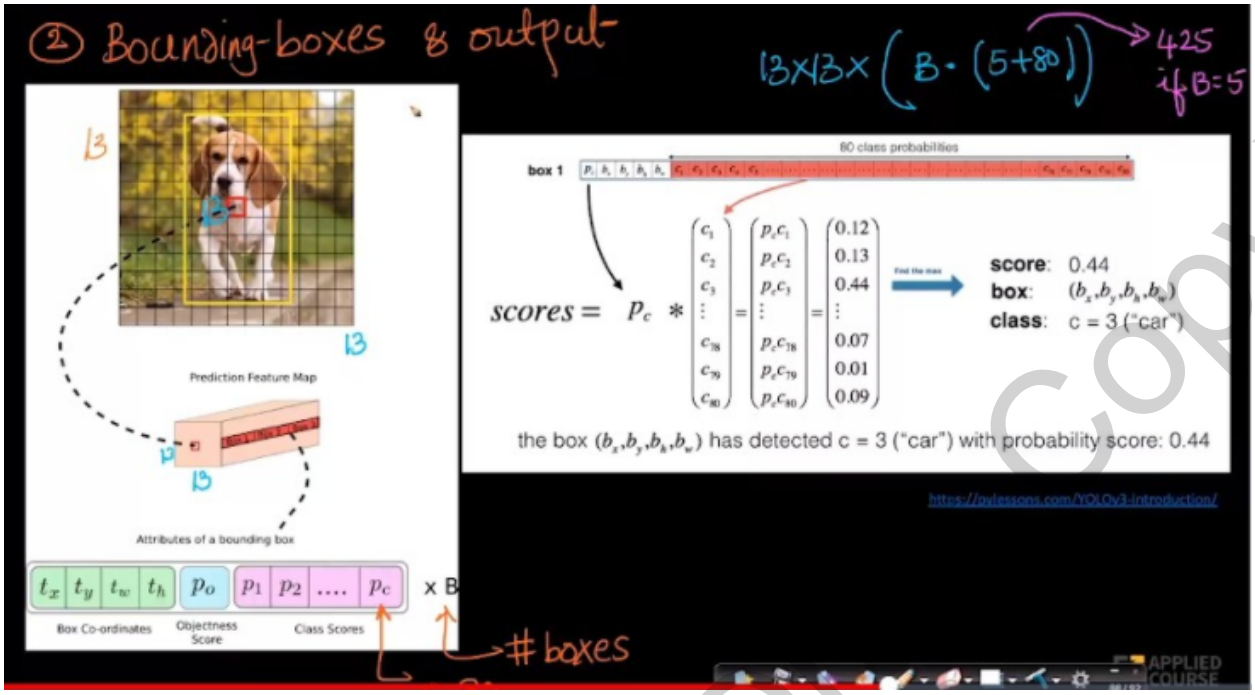
For an input image of size 416*416*3 , what will be the final output size produced by the feature extractor network ?

We count the number of times we downsample it i.e, the number of times we use the strided convolution. Because, in this stage only the size of the feature map changes. It's nothing but 5. So, the input image is reduced by 2^5 since we are dividing by half each time.

The final shape is 13*13*1024 where 1024 comes from the fact that we have used 1024 filters in the final convolutional layer.

Bounding boxes and output :

We are using a image of size 416*416*3. The output feature map produced by the feature extractor network is of size 13*13*1024. So, basically we are mapping our image of size 416*416*3 into a 13*13*1024 feature map or compressing the information contained in the image.



Timestamp : 01:27:12

It simply means, we are dividing our input image into a grid of size 13*13.

Each of those values in the grid of size 13*13 are produced from the image by applying several operations like convolution,normalization, etc. Each cell in the grid of size 13*13 has some correspondence to a part of the image. This means, a particular cell contains some information about a part of the image.

So, to encode the information about the bounding boxes, we use the 1024 dimensional vector.

So, using each cell in the 13*13 grid as a centre we generate several anchor boxes. These anchor boxes contain some information right ?

What are those ?

tx : x coordinate of the centre.
ty : y coordinate of the centre.

tw : width of the anchor box.
th : height of the anchor box.
p0 : objectness score. It gives the confidence of any object present in the anchor box. If it's zero, then there is no object contained in the anchor box which is present in the training data.
p1,p2,..pc : P(c10) = p0 * p10 (It gives the conditional probability of an object of class 10 being present in the anchor given any object is present. Because, if no object is present, then there is no use of calculating pc for any c.

There are B number of anchor boxes generated by considering a cell in the 13*13 grid as centre. How can we generate B such ? By simply varying the width and height i.e, tw and th.

So, for each anchor box, we have 5+c values where c is 80 here. So, there are a total of 85 values for each of those B boxes.

With the same centre and varying tw and th , we can generate multiple anchor boxes. This means it can cover multiple objects. So, only we have B anchor boxes.

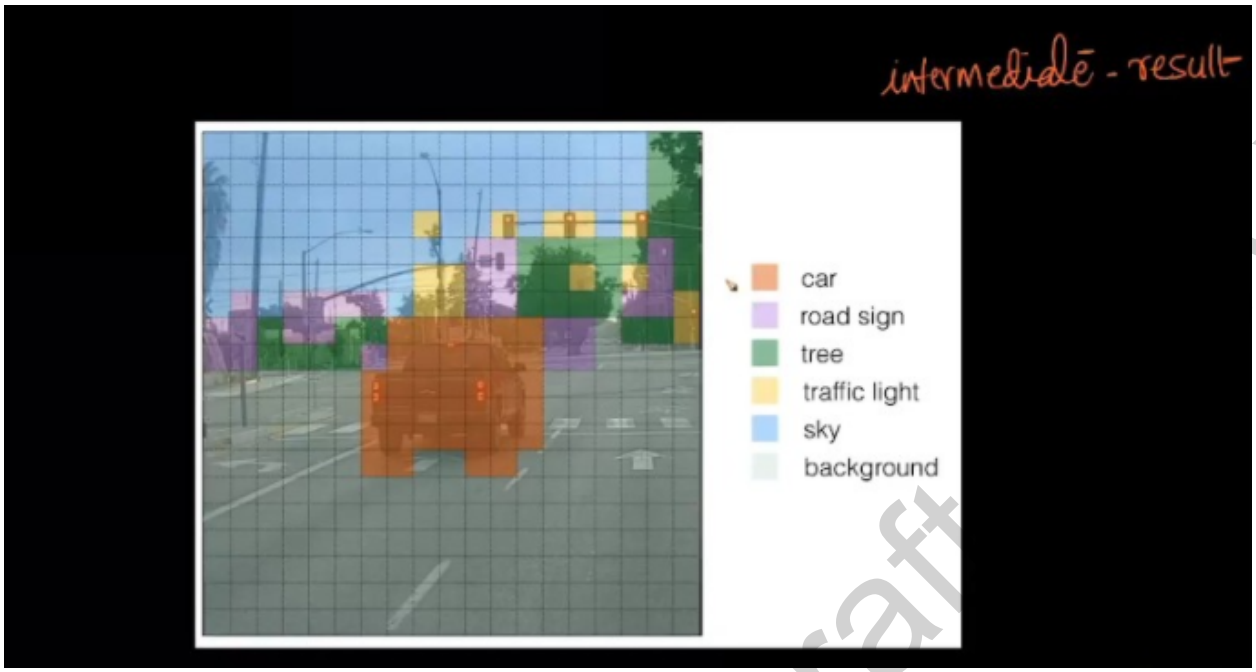
The difference between a bounding box and an anchor box is that the bounding box is what the ground truth depicts and the anchor box is what we generate as a prediction.

General formula : 13*13*(B*(5+c)) where c is the number of classes.

If B is 5 and c is 80, then it becomes 13*13*425.

But our final output shape was 13*13*1024. So, to transform this to 13*13*425, we use a 1*1 convolution with 425 filters.

Let's visualize the intermediate result.

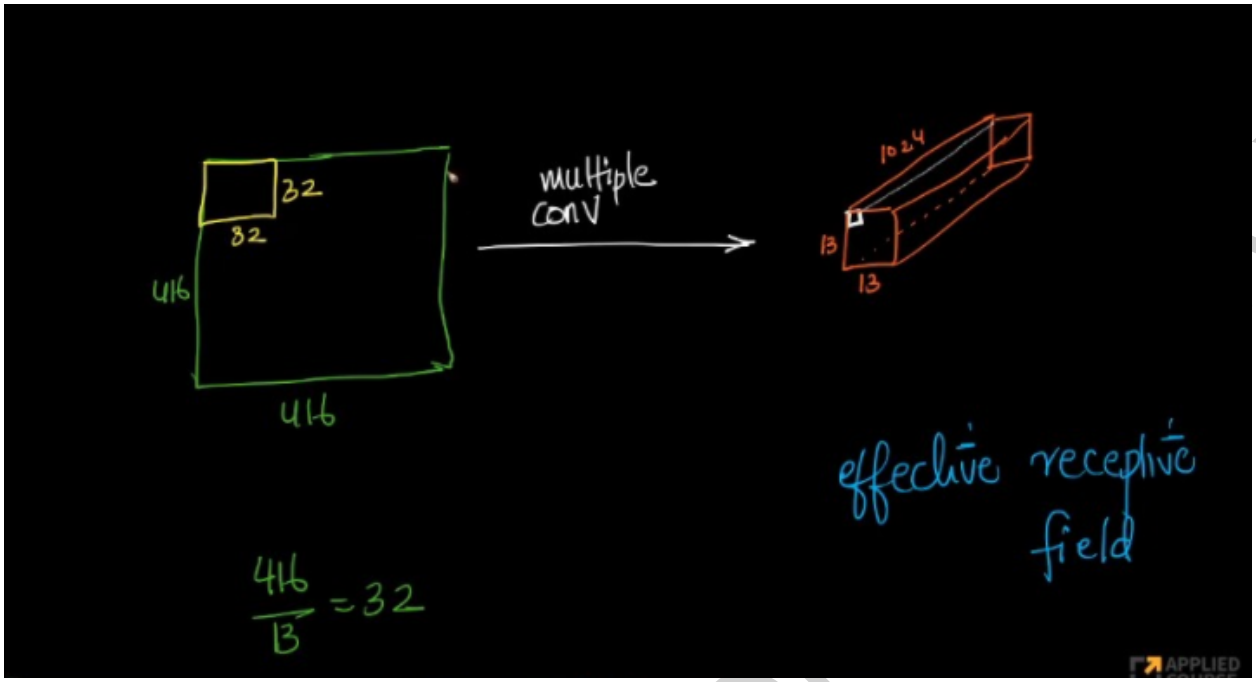


Timestamp : 01:41:33

Please refer to the color coding given above.

So, for each cell, we have multiple anchor boxes right ? So, each anchor box predicts an object contained in it. There will be B predictions since we have B anchor boxes. Out of those B predictions, we take the maximum one. Then, we color it based on the class we got. So, the model was able to correctly predict the car in the image. Also, the same applies for other objects too.

69.2 Object Detection YOLO V3



Timestamp : 04:32

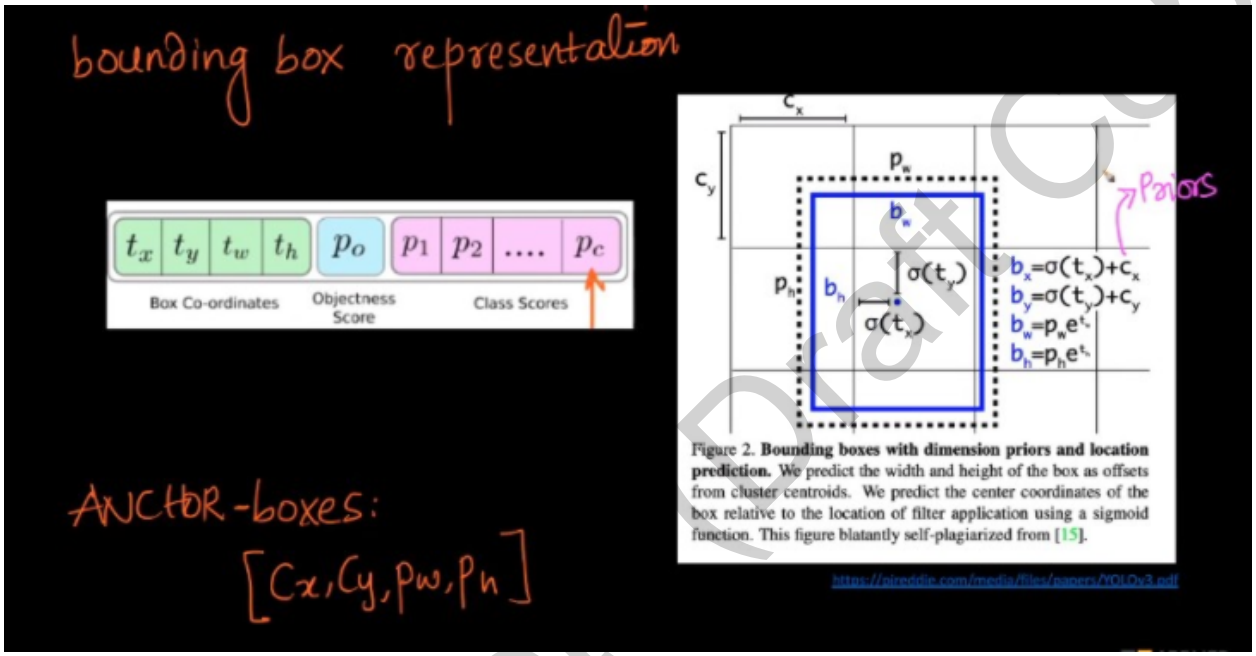
We already seen that, the final output tensor we got is of shape 13*13*1024. These are produced by applying several convolutions and other operations in the input image. Let's look at it closer.

Each of those 1024 dimensional vector in the 13*13*1024 tensor has some correspondence in the input image. Basically, the first 1024 dimensional vector are actually produced by applying several convolutions and other operations in the first grid of size 32*32 in the input image. This means, the first 1024 dimensional vector is the compressed version of 32*32*3 tensor (We have RGB channels). So, any object which is present in the 32*32 grid can be identified by using the first 1024dimensional vector in the 13*13*1024 tensor. This is called an effective receptive field.

We already discussed the effective receptive field. It simply means, the receptive field in the image which corresponds to the output feature map.

Bounding box representation :

Predicting the actual bounding box co-ordinate is quite hard. This is because the output of a neural network is not bounded. We already saw this while learning logistic regression. The logits are not bounded. So, we applied a sigmoid function to it, so that the range is bounded in an interval. Similarly, we are going to perform the same here.



Timestamp : 07:50

For this, first we need to understand the concept of anchor boxes.

Anchor boxes : They are pre-defined boxes specified as part of the algorithm. Bounding boxes represent the actual truth. For example, while creating the object detection dataset, we want to label each object in the image by a rectangle. This rectangle is called a bounding box. It's the actual one. Anchor boxes are defined so that we can use it to predict the bounding box.

The idea is simple. We already discussed how we would generate anchor boxes. So, we first divide the image into 13*13 grid. Then, using each cell in the grid of size 13*13 as centre i.e, cx and cy we generate multiple anchor boxes by varying tw and th. We do this for every cell in the grid. We

also have the original coordinates of the bounding box which covers the object (It's part of the dataset). Our task is to predict the bounding box or do some changes to the anchor boxes such that the anchor box serves as the prediction for the bounding box.

Let's assume that bx and by are the centre coordinates of the bounding box.

Let's also assume that bw and bh are the width and height of the bounding box.

So,

bx = sigmoid(tx) + cx
by = sigmoid(ty) + cy
bw = pw*e^tw
bh = ph*e^th.

Here cx,cy are the centre coordinates of the anchor box. Pw and ph are the width and height of the anchor box.

The reason why we are exponentiating and applying the sigmoid function is little bit involved.

$$\left(\frac{\frac{x_b - x_a}{w_a} - \mu_x}{\sigma_x}, \frac{\frac{y_b - y_a}{h_a} - \mu_y}{\sigma_y}, \frac{\log \frac{w_b}{w_a} - \mu_w}{\sigma_w}, \frac{\log \frac{h_b}{h_a} - \mu_h}{\sigma_h} \right),$$

Here xa,ya are the centre coordinates of the anchor box A.
Here xb,yb are the centre coordinates of the bounding box B.

We are trying to transform the anchor box A to bounding box B. So, we are finding the offsets. The reason why exponentiate is because we are using the log transformation.

$$\mu_x = \mu_y = \mu_w = \mu_h = 0, \sigma_x = \sigma_y = 0.1$$

$$\sigma_w = \sigma_h = 0.2.$$

Given varying positions and sizes of different boxes in the dataset, we can apply transformations to those relative positions and sizes that may lead to more uniformly distributed offsets that are easier to fit.

How to design anchor boxes or how to generate them ?

So, we have a set of bounding boxes in our training data which are labelled. In this collection of bounding boxes, we run k-means clustering with k=5. This means, we have generated 5 bounding boxes per cell.

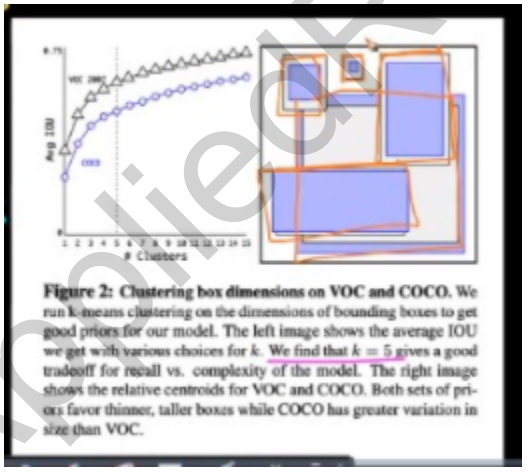


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

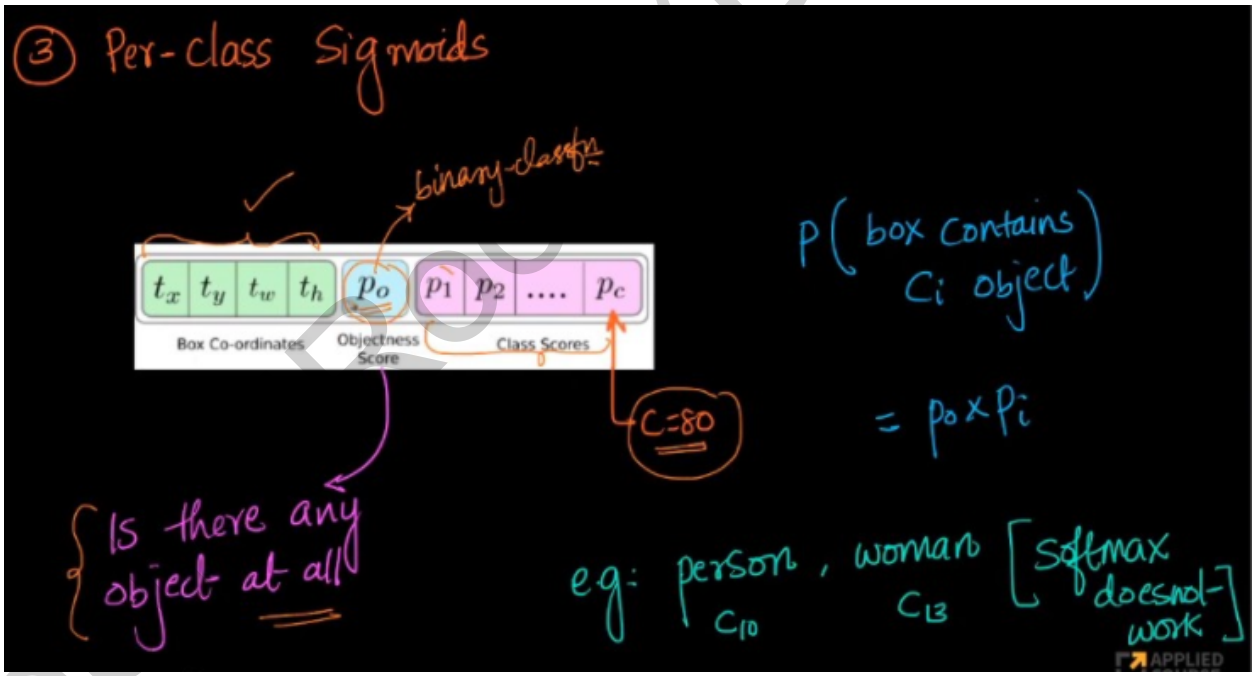
Timestamp : 15:13

The authors chose the value for k as 5 since it gave a good tradeoff for recall vs complexity of the model.

These are nothing but the prior for bounding boxes.

Per class sigmoids :

Objectness score (p0) : It is nothing but the probability of finding an object(It could be any object present in the training data) given an anchor box A .
It's a binary classification problem. Because, either the anchor box A contains an object which is present in the training data or not. So, we can use a binary classifier for this.



Timestamp : 18:17

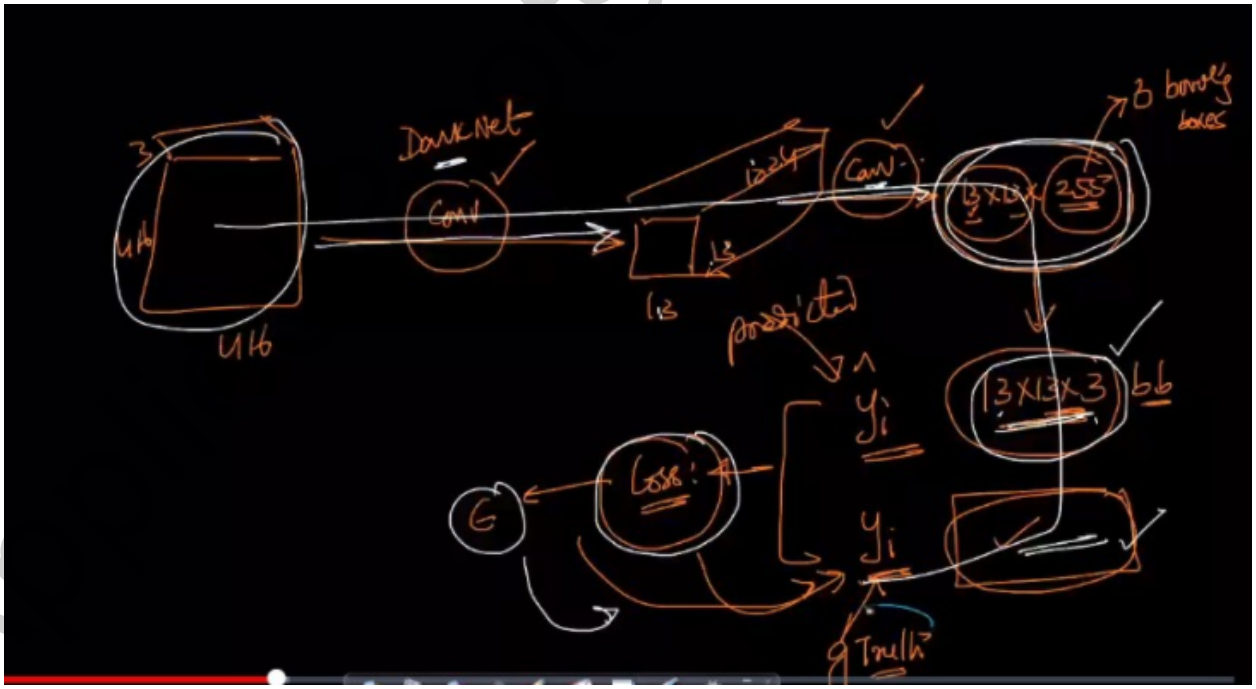
What does p1,..pc represent ?

P(anchor box contain object of class Ci) = p0 * pi (By definition).

There are 80 classes in the COCO dataset. Since we have multiple classes, it's natural to define a softmax classifier. But there is a catch. Let's look at it closer. The problem here is that a single object can actually belong to multiple classes. For example, an object can be both a person and a woman. This can certainly be true. By using softmax, we can't model this. Why ? The summation of the softmax vector is always 1. This means, if the object is a woman, then the corresponding cell in that vector would be close to 1 and others will be close to zero. But, this object is also a person. We also want a higher value for that corresponding cell. But, the softmax classifier doesn't help.

So, to tackle this problem, we are going to learn a binary classifier for each of those classes. This solves the problem. A binary classifier could be a logistic regressor. This means, we are going to learn c such models.

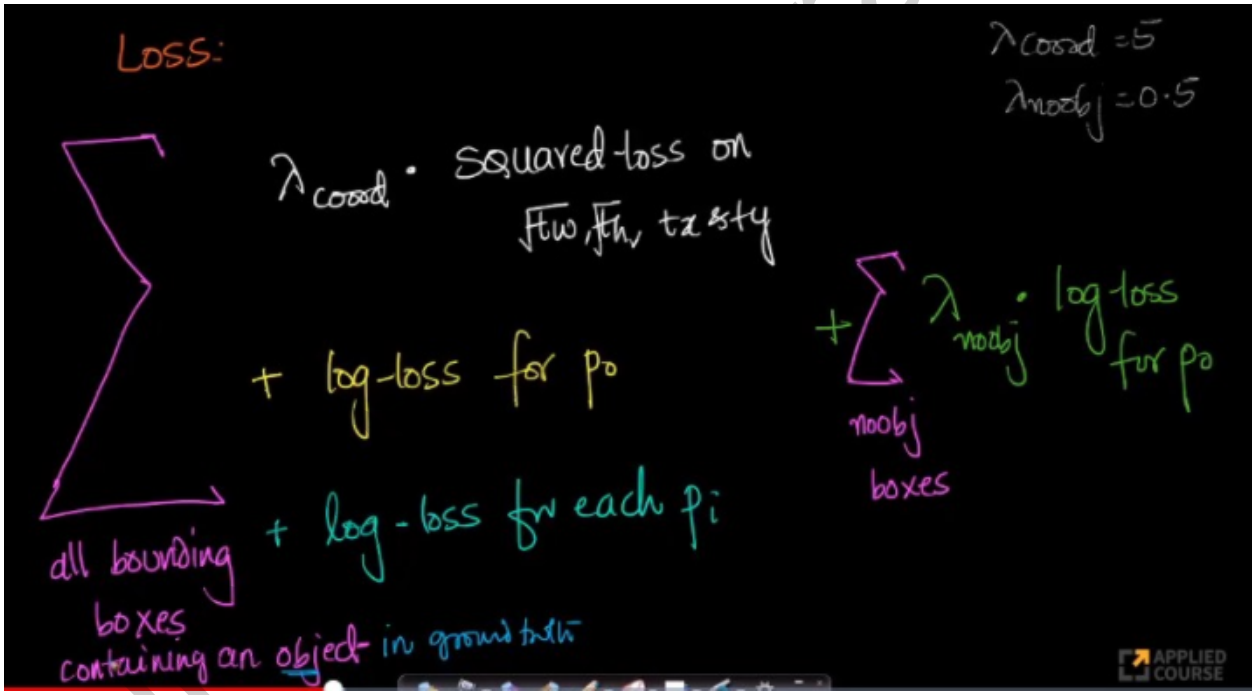
Loss function :



Timestamp : 28:44

We already discussed the flow of inputs in the network until the output is produced. For optimization, we need to have ground truth and the predictions. So, we can compute the loss and backpropagate. This will tune the parameters in the network such that the loss is as minimum as possible. In the above example, we are considering 3 anchor boxes per cell. So, the output tensor is of shape 13*13*255 (involving 80 classes). We also have the actual y.Then, we compute the loss and backpropagate. If p0 is small or close to zero, then we can ignore that prediction since it means there is no object or the confidence level of finding an object in that specified region is very low.

Now, what is the loss here ?



Timestamp : 28:48

Take all the bounding boxes containing an object in ground truth.

Then, use mse loss on sqrt(tw), sqrt(th), tx and ty with a weighting parameter lambda_coord.

For p0, we can use log-loss.

For each of those pi, we are learning an independent binary classifier as discussed earlier. So, we use log-loss for each of those models.

Then, take all the bounding boxes containing no object in it.

Use log-loss for p0 with a weighting parameter lambda_noobj. If there is no object, there is no need to predict the anchor boxes or class probabilities for it. So, we ignore them.

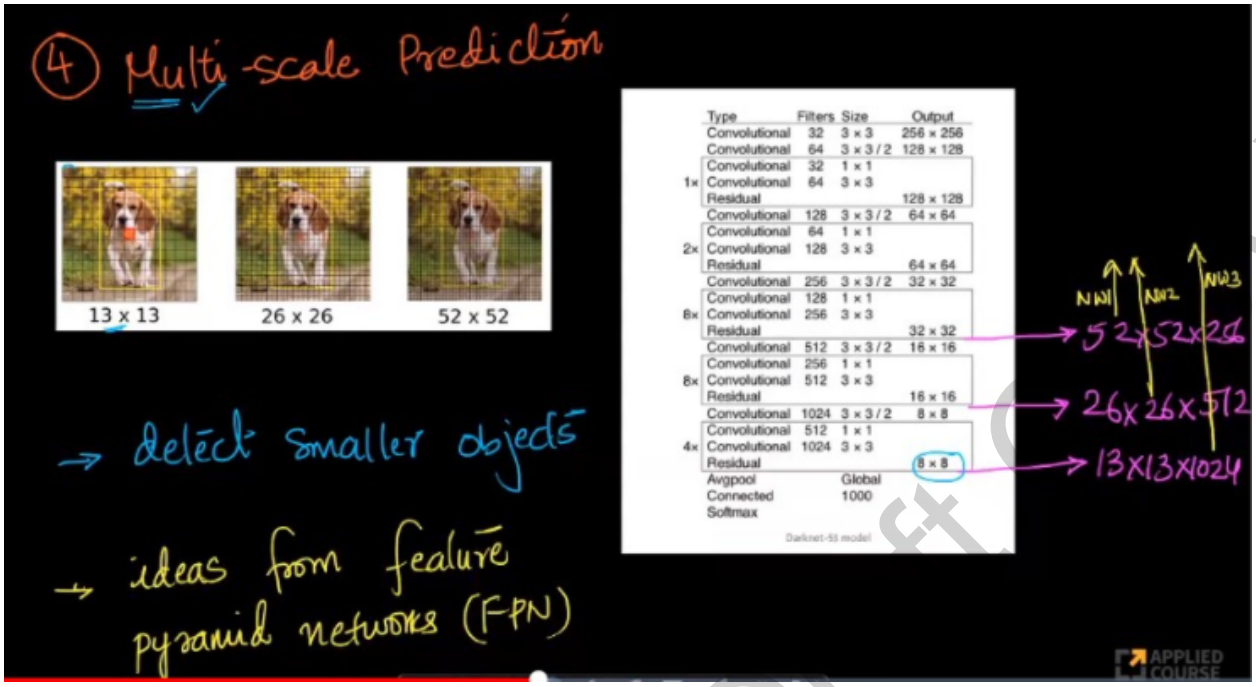
Now, we need to understand lambda_coord and lambda_noobj.

The idea of using them is simple. The variables we are going to predict are not in the same range. For example, p0 and tx are not in the same range. P0 lies in the interval [0,1] and tx in the interval [-inf,+inf]. So, if we compute the individual losses as we did in the above image, the magnitude of individual losses could vary. So, we are inducing a bias here. So, the model may only try to minimize the first part of loss and ignores the second one due to it's less magnitude. So, we can have a scaling factor which controls it. This is similar to the lambda parameter in regularization.

The best value they got for lambda_coord is 5 and for lambda_noobj to be 0.5

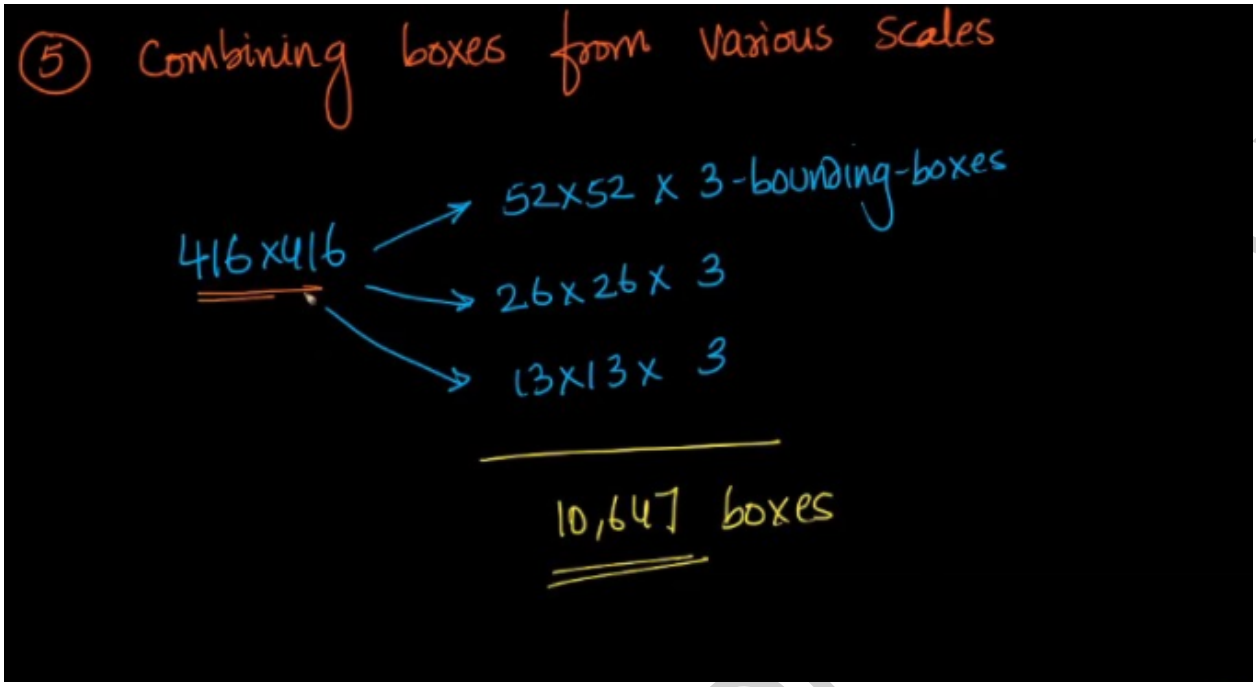
Additional Note : The darknet architecture we are using in the YOLO-V3 model is pre trained and the weights for this network are not randomly initialized. The other layers which we use to generate the final output tensor are only initialized by us.

Multi scale Prediction :



Timestamp : 45:01

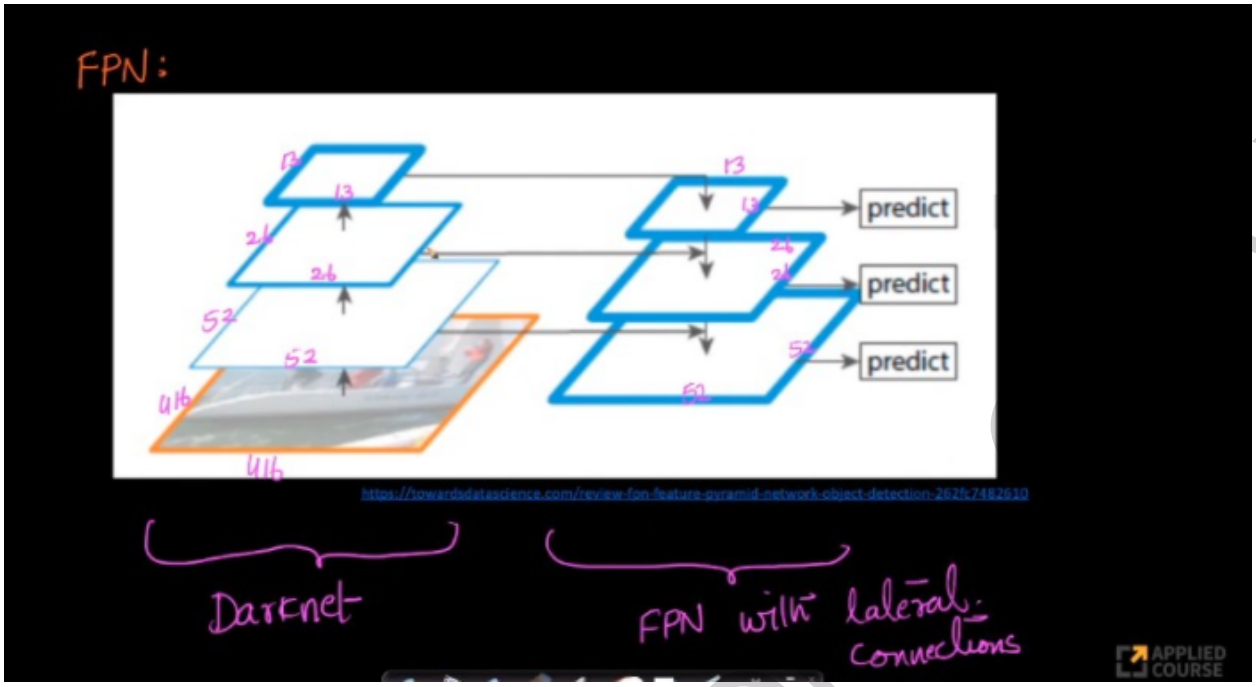
The output tensor shape is 13*13*255. This means, we are transforming our image into a grid of size 13*13. It can detect large objects. In an image, typically the number of objects with large size is lesser than the number of objects with smaller size. This means, we need more number of anchor boxes for an image containing smaller objects and less number of anchor boxes for an image containing larger objects. If you think about it, we can sample less number of anchor boxes in a 13*13 grid when compared to a 26*26 grid. This means, we can transform our image to a 26*26 grid so that we can sample more anchor boxes and also it can help detect smaller objects since the cell size is small. This is called multi-scale prediction. We are actually using the scale of the grid to detect small or larger objects. This idea is actually derived from feature pyramid networks.



Timestamp : 48:48

From the above image we can see that we are generating anchor boxes at three different scales. We can calculate the total number of anchor boxes across all the scales. The maximum number of anchor boxes we can have is 10,647. But for some of those, the value of p0 may be small or close to zero. So, we can safely ignore them.

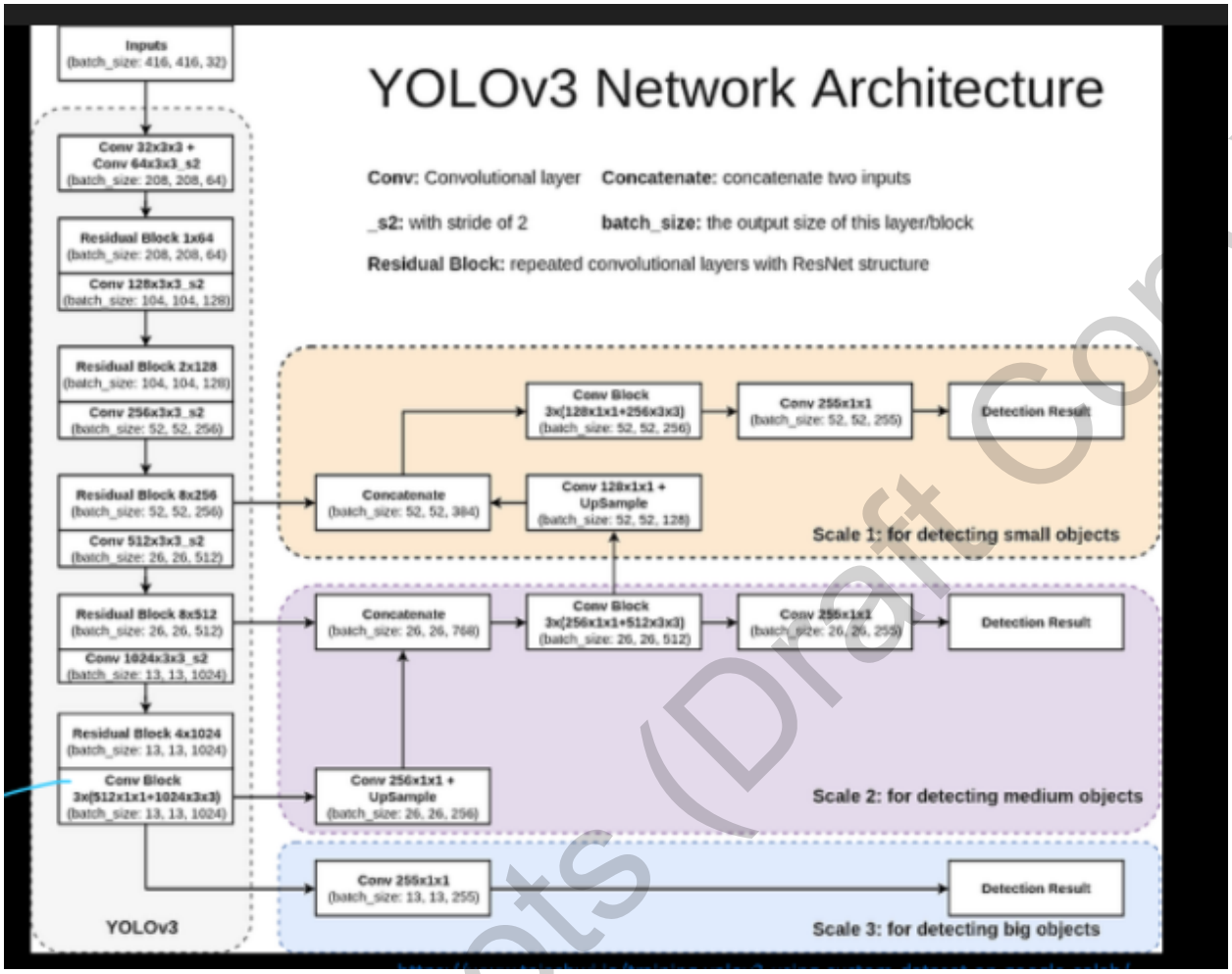
Feature pyramid networks :



Timestamp : 53:10

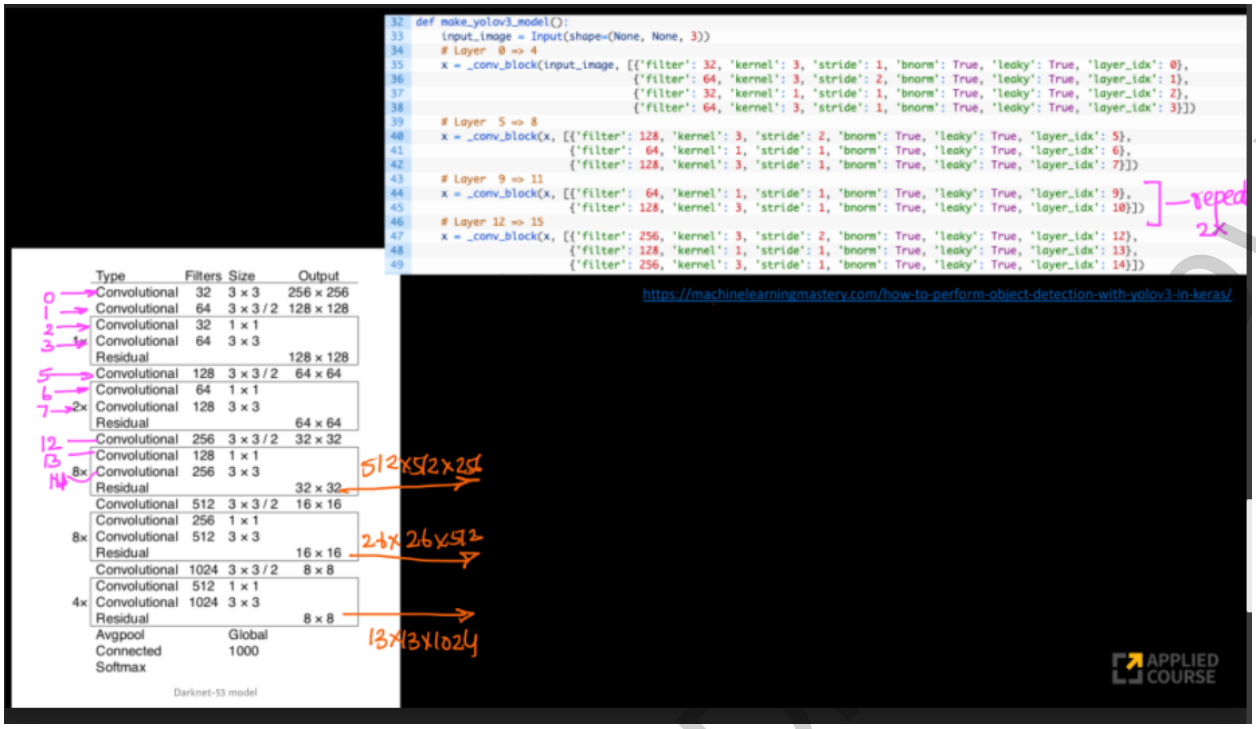
Traditionally, we predict anchor boxes at multiple scales as discussed earlier. While generating multiple scales, we didn't use any information across the scales. For example, while generating the 26*26*2555 tensor, we didn't use any information from the 13*13*255 tensor. Because, there may be some information present in the 13*13*255 tensor which may be useful to generate the 26*26*255 tensor. Another idea is that a large object is nothing but a composite of smaller objects. So, we can use this idea here. So, in fpn we have those lateral connections as mentioned in the second figure which actually helps us in using the information.

YOLO-V3 Network Architecture :



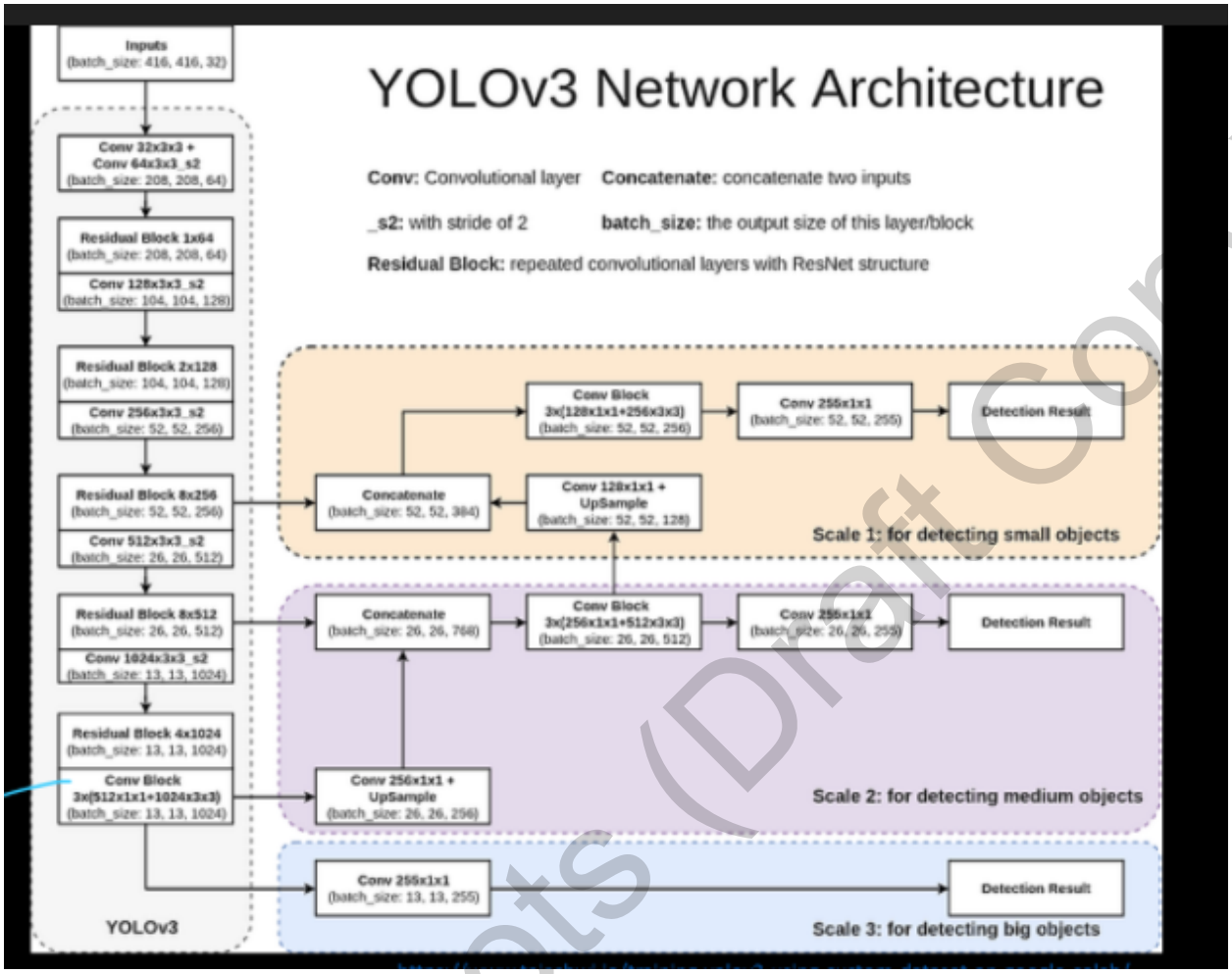
Timestamp : 54:09

The leftmost part is already discussed, which is the DarkNet architecture.



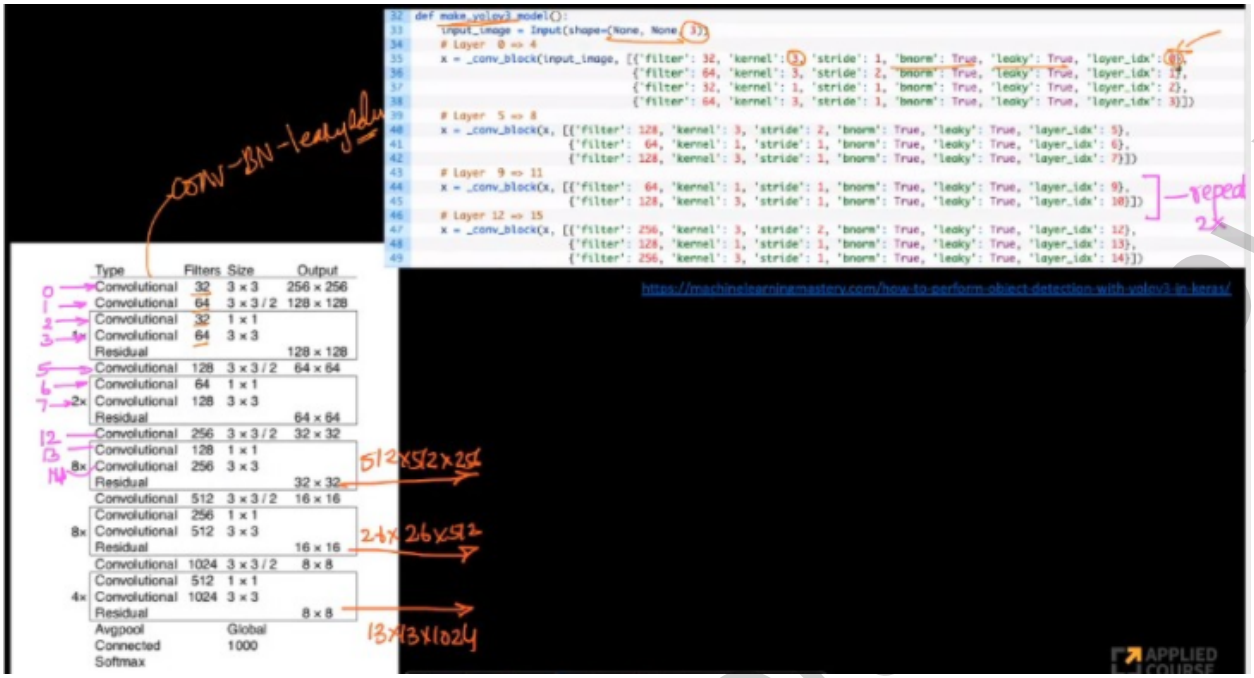
Timestamp : 58:03

The reason why we used 3*3 kernels rather than only relying on 1*1 is that, the 3*3 has larger receptive field than a 1*1 kernel and also it uses the information from the neighbourhood pixels more than a 1*1 kernel.



We also want to do multi scale predictions as we discussed earlier. So, to do this, we perform Upsampling operation rather than downsampling it. We can see this from the above block diagram which is colored with purple. Also, we need to use the concept of a feature pyramid here i.e, using the lateral connections. There is a connection between the first block and the second block. This means, we are using the features learnt for detecting medium objects to detect smaller objects.

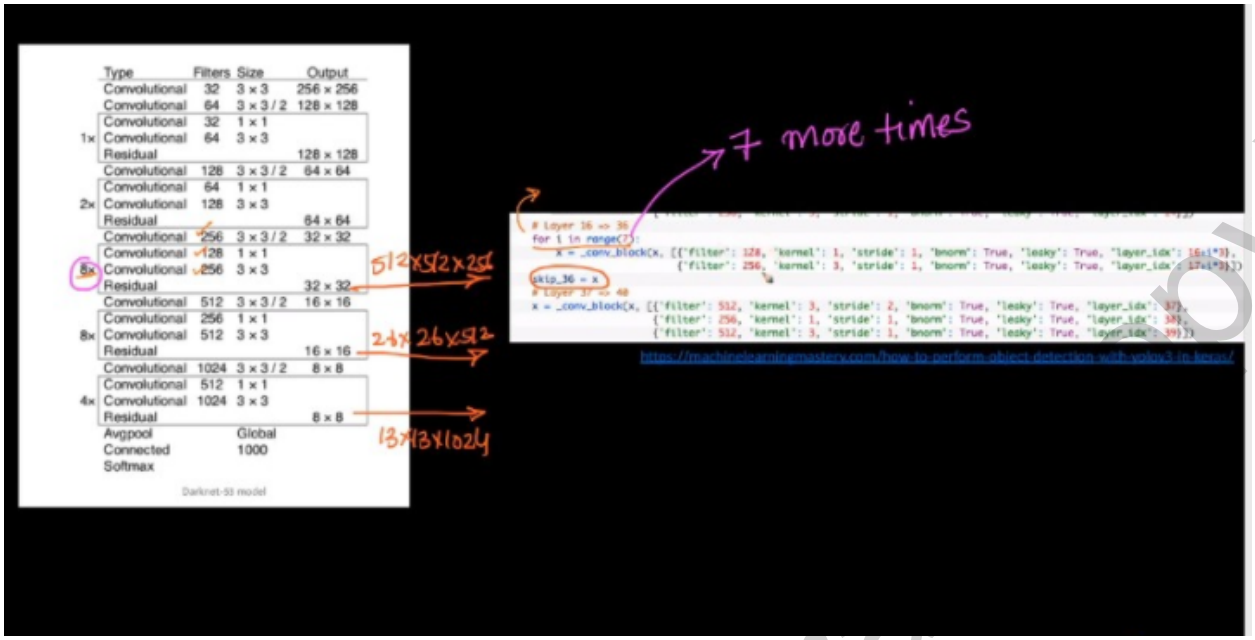
Concatenate layer simply concatenates on the feature dimension i.e the last dimensions. For example concatenation of 26*26*512 ans 26*26*256 yields a 26*26*768 tensor i.e we are concatenating on the last axis. We know how to upsample. For this, we use transpose convolution. If we follow the direction of the arrow, we can understand the flow.



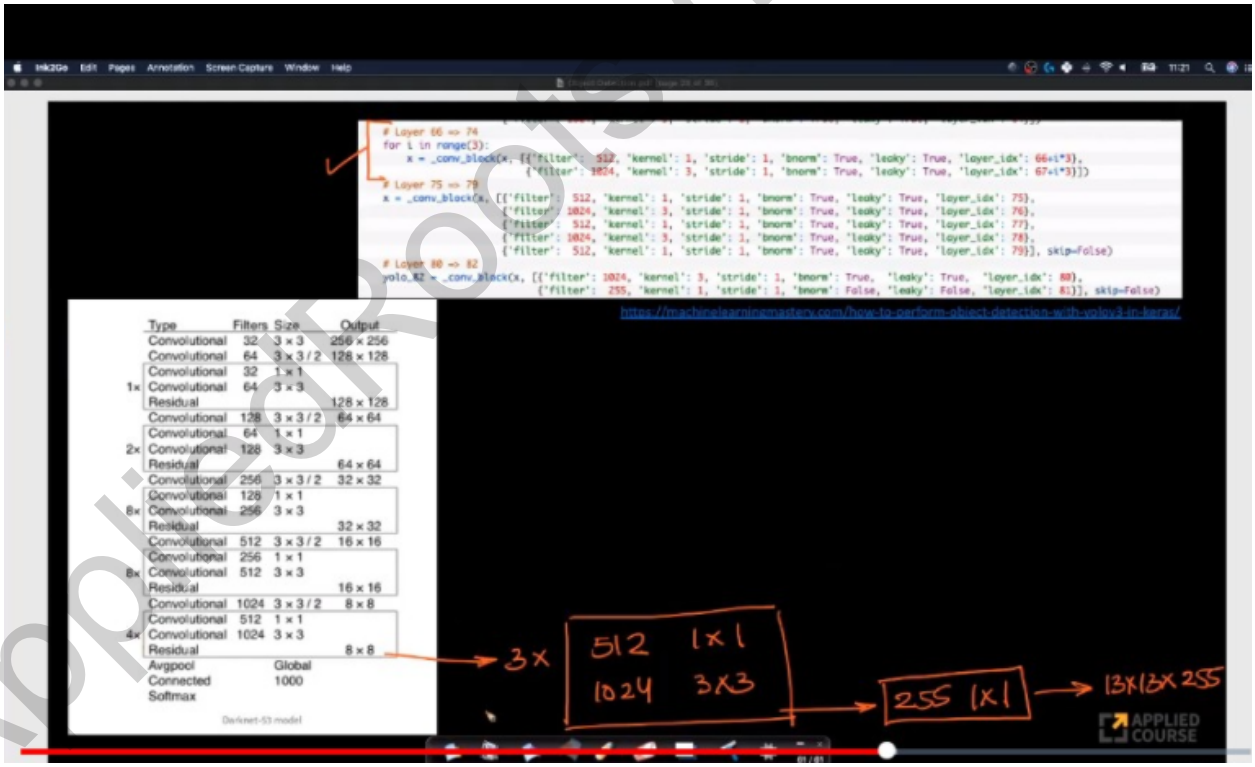
Timestamp : 01:06:21

The code is actually easier to understand. The architecture is specified in the left region. We are just using keras to define the architecture. It's also intuitive.

Some blocks are repeated like 8 times or i times. For this, we can use the conditional loop statement in python.



Timestamp : 01:09:41



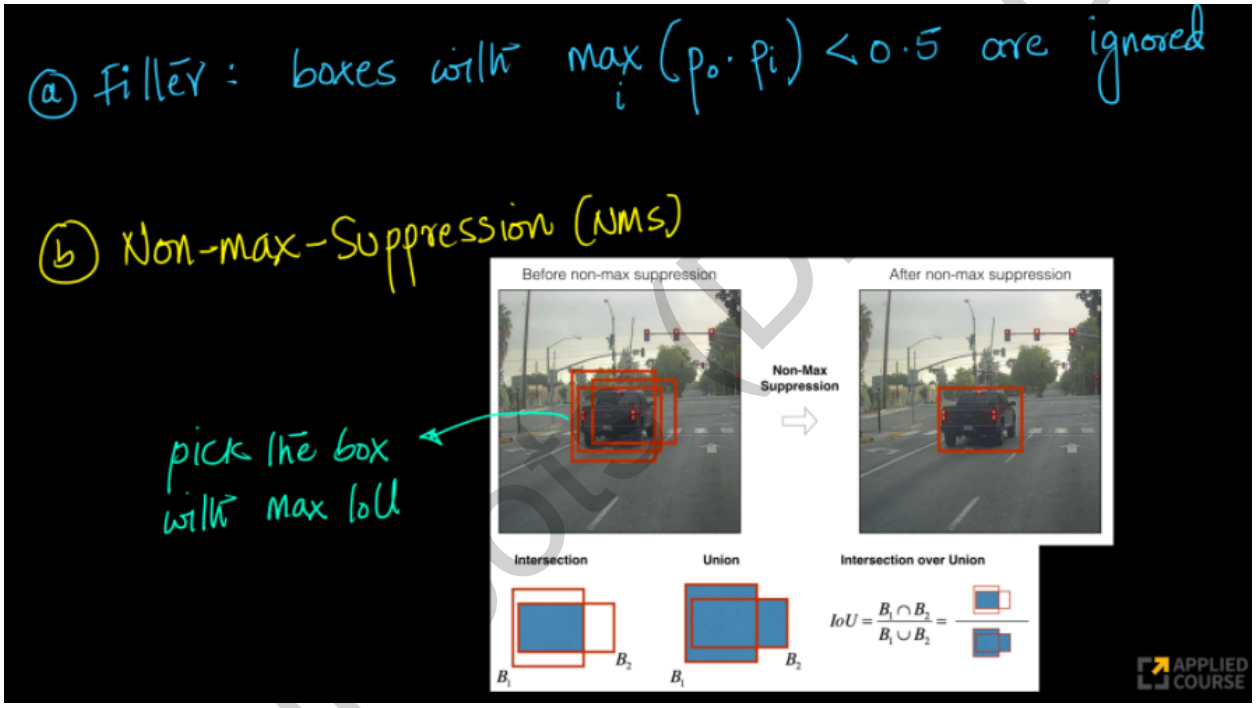
Timestamp : 01:12:40

It's self explanatory. Skip = False means there is no skip connection or residual connection.

Filter :

Boxes with max (po*pi) < 0.5 are ignored for all i.

Non-max suppression :



Timestamp : 01:16:10

There may be a case where multiple anchor boxes predict the same object. This means, the pi for each of those anchor boxes is maximum. So, they all point to the same object. But ultimately we want only one anchor box for an object.

How to choose one among them ?

Non-maximum Suppression Algorithm

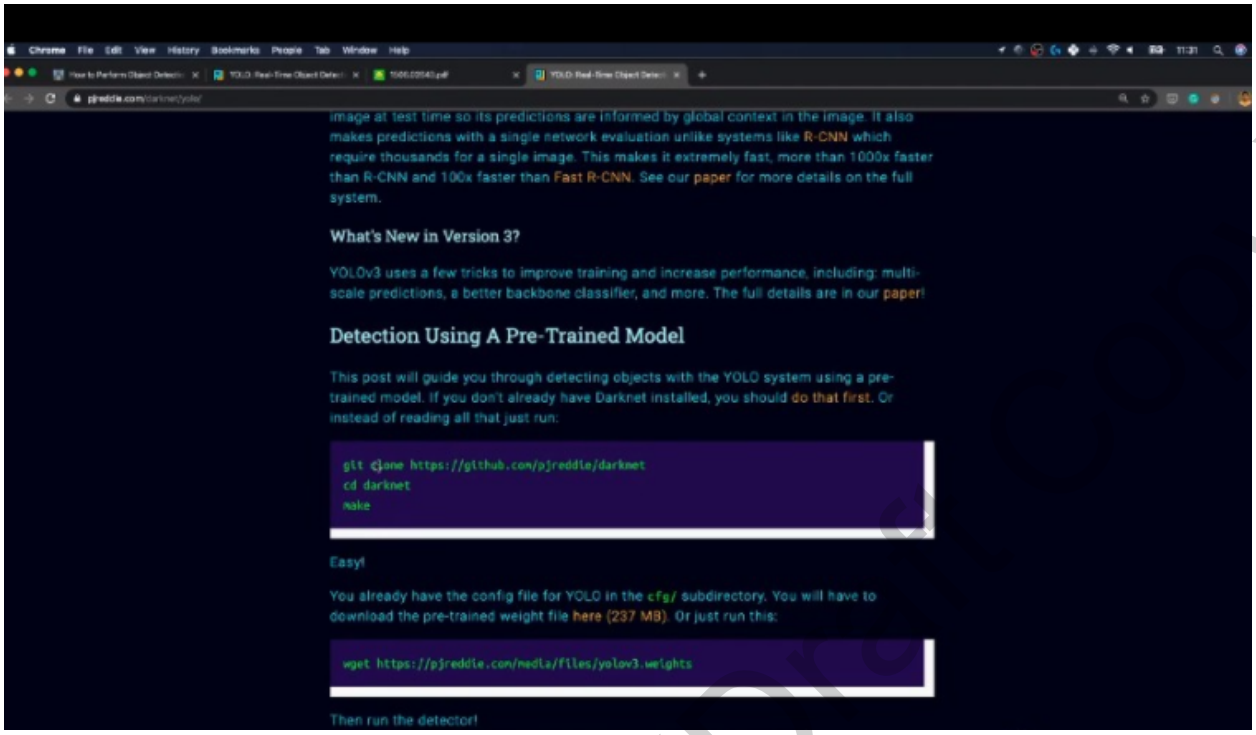
Input: A list of Proposal boxes B, corresponding confidence scores S and overlap threshold N.

Output: A list of filtered proposals D.

Algorithm:

1.) Select the proposal with the highest confidence score, remove it from B and add it to the final proposal list D. (Initially D is empty).
2.) Now compare this proposal with all the proposals — calculate the IOU (Intersection over Union) of this proposal with every other proposal. If the IOU is greater than the threshold N, remove that proposal from B.
3.) Again take the proposal with the highest confidence from the remaining proposals in B and remove it from B and add it to D.
4.) Once again calculate the IOU of this proposal with all the proposals in B and eliminate the boxes which have a higher IOU than threshold.
5.) This process is repeated until there are no more proposals left in B.

Detection using a pre-trained model.



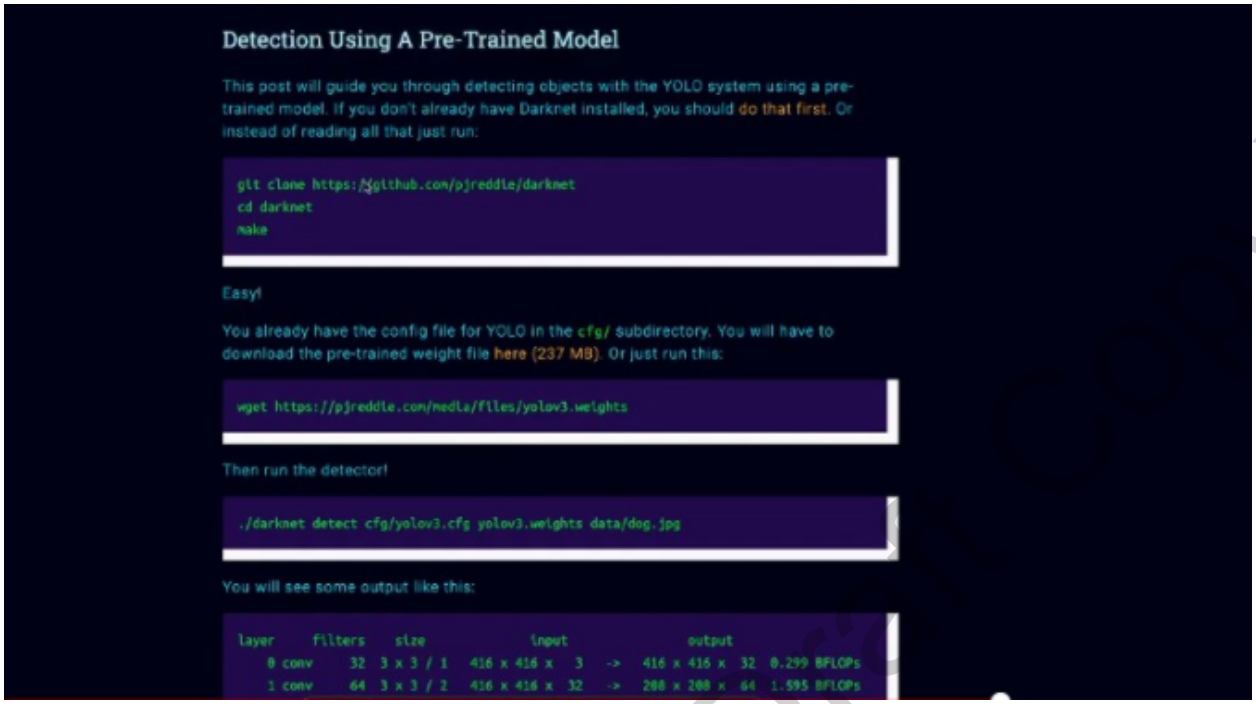
Timestamp : 01:23:02

First, we clone the repository from github where it holds the necessary code files.

Then, we change the current directory to darknet by using cd command.

Then, we use the command “make” to compile it.

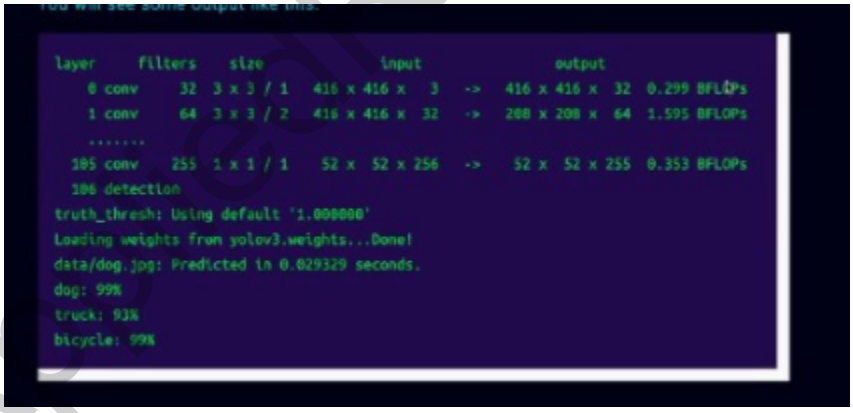
Using wget command, we can download the weights of the yolo-v3 model.



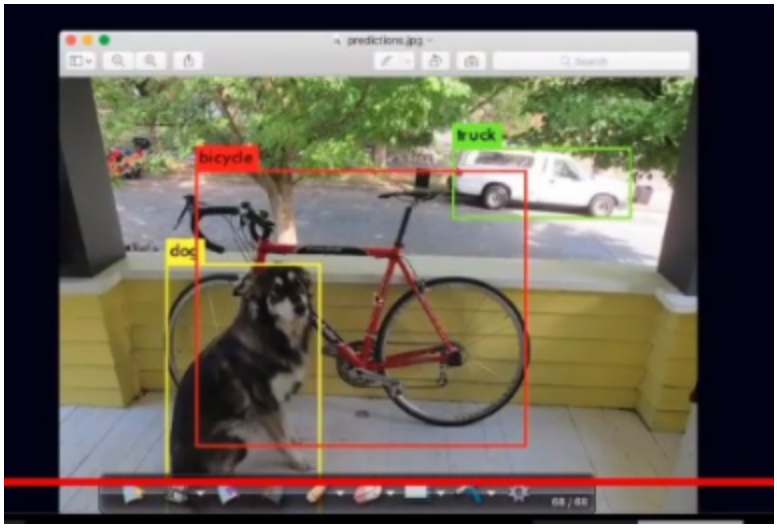
Timestamp : 01:23:27

Then, we can run the detector on any image. For this example, we are using the dog.jpg file. We are providing the configuration files which holds the information about the architecture and the weights.

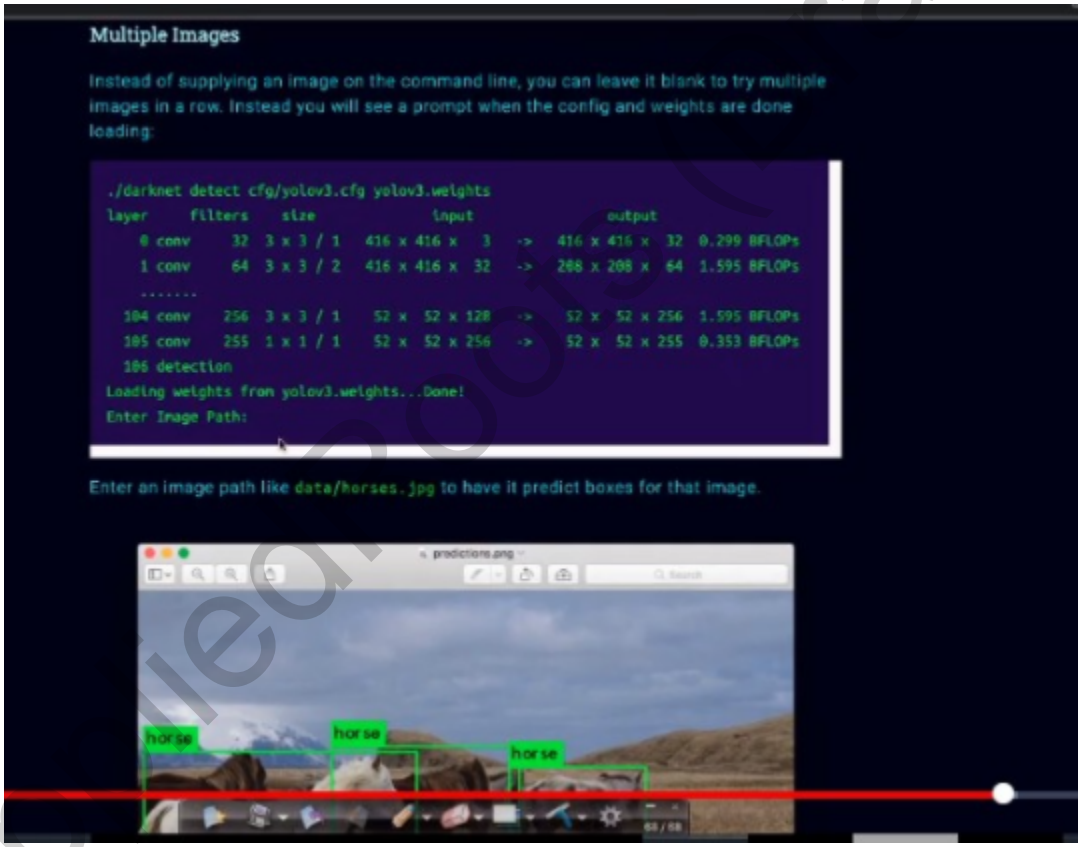
Output :



Timestamp : 01:24:54



Timestamp : 01:24:59



Timestamp : 01:26:37

To deal with multiple images, we can simply type the path in the console and get the predictions.

We can change the detection threshold if we want. Fordefault, any object which has p0>0.25 will have its place. But, we can modify it.

Timestamp: 01:27:32

If we have some restrictions on computing resources, then we can use Tiny Yolo-v3.

Timestamp : 01:27:46

We can also perform real-time detection on a webcam.

Timestamp : 01:28:38

For better real-time detection, we need to have a good GPU.

For additional references, please refer to [this]