**Lesson 3:**

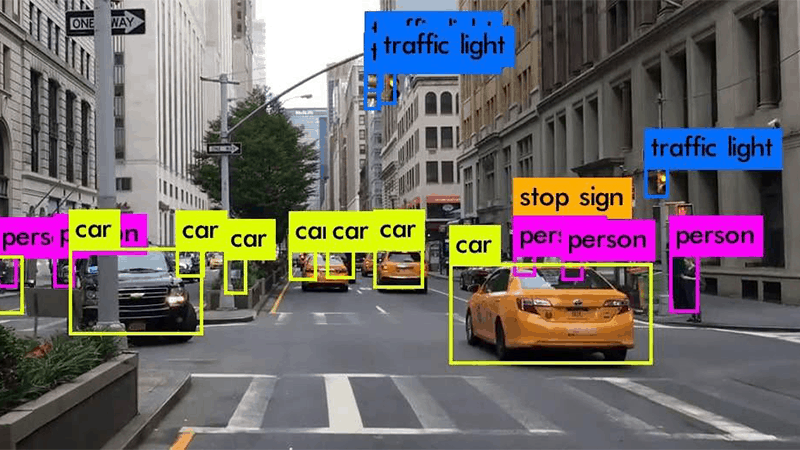
***Classifiers*** are files or references we use to identify objects in images.

The act of identifying which kind of object there is in an image is ***classification***.

The act of identifying the location of an object in an image is ***localization***.

The rectangle that exists in the object location is called a ***bounding box***.

Example:



**Lesson 4:**

YOLO (You Only look once)

Is used the method of object detection of ***single pass***, which use a convolutional network as a ***feature*** extractor.

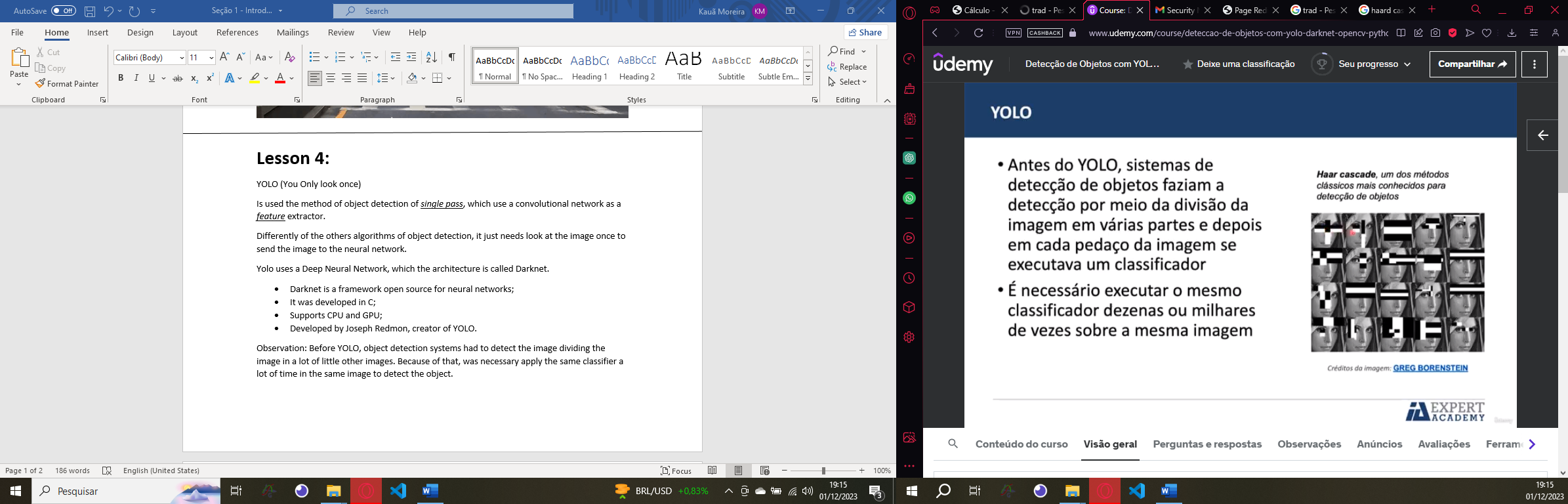
Differently of the others algorithms of object detection, it just needs look at the image once to send the image to the neural network.

Yolo uses a Deep Neural Network, which the architecture is called Darknet.

* Darknet is a framework open source for neural networks;
* It was developed in C;
* Supports CPU and GPU;
* Developed by Joseph Redmon, creator of YOLO.

Observation: Before YOLO, object detection systems had to detect the image dividing the image in a lot of little other images. Because of that, was necessary apply the same classifier a lot of time in the same image to detect the object.

Example:



Therefore, the advantage of TOLO is the fact that it classifies the object by looking only once, without having to divide the images into other small images.

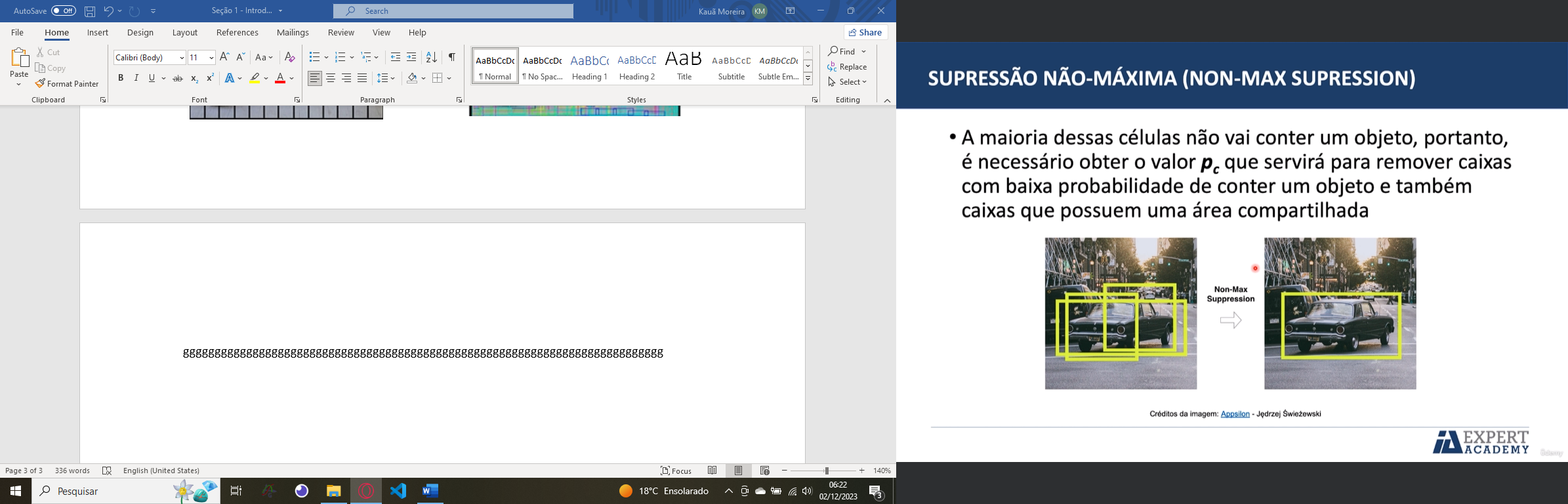
**Lesson 5:**

How does YOLO works?

* The algorithm divides the image in a grid of SxS cells, normally it is used 13x13=169 cells;
* Each cell is responsible for make de prediction of 5 bounding boxes in different positions It means: 5.169=845 bounding boxes;
* The YOLO returns a confidence value: How much the algorithm is sure if that bounding box have an object or not;
* For each box, the cells also do the prediction of a class for all the object classes possible;
* The most of those boxes will have a confidence value extremally low. Because of that, the algorithm considers just boxes with 30% of confidence or more. This filtering technique is called as ***threshold***.

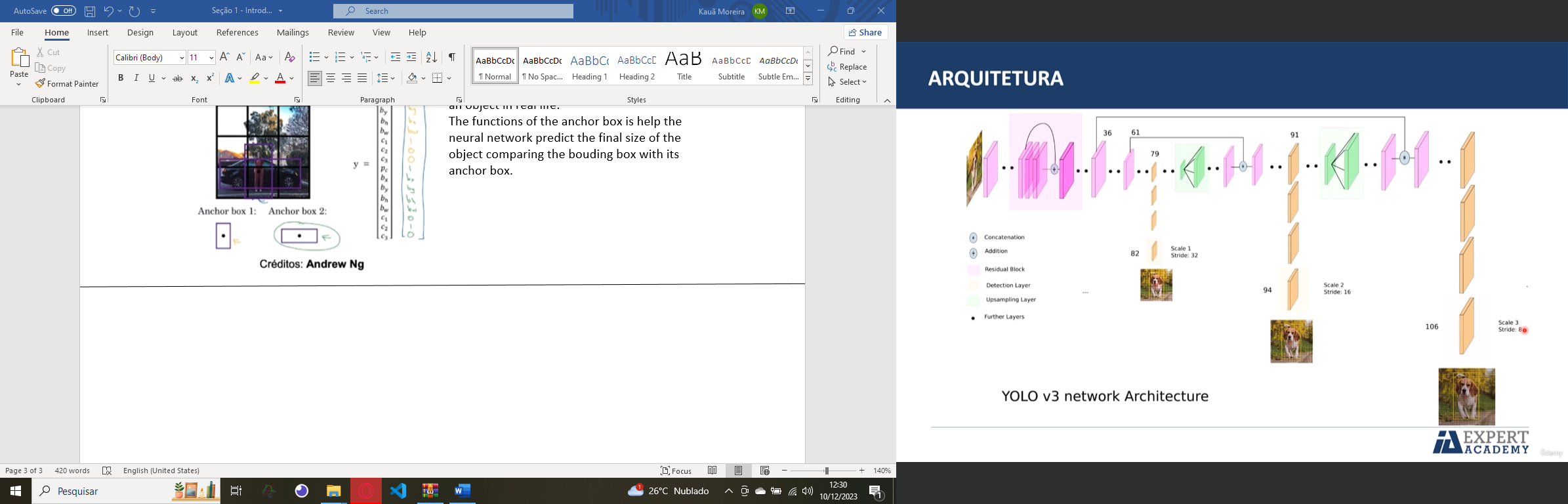
|  |  |
| --- | --- |
| 169(13x13) cells processing: | Bounding boxes processing: |
|  |  |

The ***Non-max suppresion*** is a technic used in YOLO algorithm in situations that we have a lot o bouding boxes above themselfes, the YOLO gets the bounding boxe with biggest confidence value and define this as the “winner bouding boxe”.



|  |  |
| --- | --- |
|  | ***Anchor boxes*** are boxes with defined size(width and height) near the real size of an object in real life.  The functions of the anchor box is help the neural network predict the final size of the object comparing the bouding box with its anchor box. |

**Lesson 7:**



Yolo have three output layers that give us three different responses layer\_outputs[0], layer\_outputs[1], layer\_outputs[2].

**Lesson 9 - 13:**

***CUDA ("Compute Unified Device Architecture")*** is a parallel computing platform and programming model developed by NVIDIA. It allows developers to use NVIDIA GPUs (Graphics Processing Units) for general-purpose processing, not just for graphics-related tasks.

At its core, CUDA provides a framework and set of tools that enable developers to harness the immense computational power of GPUs for a wide range of tasks beyond traditional graphics rendering. It allows for highly parallel computations, leveraging the thousands of cores available in modern GPUs.

Linux commands in sheel:

* “ls {selectedFolder}” show to you the folders and files inside the selectedFolder;
* “!./darknet detect cfg/yolov4.cfg yolov4.weights {imageDirectory}” Try identify the image in the imageDirectory;
* “!cat data/coco.names” show the objects you can find the images by Yolo;
* !cp {fileDirectory} {folderDirectory} Copy what is in the fileDirectory to the folderDirectory.

**Lesson 22:**

To use the YoloV4 with OpenCV, it is necessary just three files: coco.names, yolov4.weights and yolov4.cfg!

coco.names: The file with the label name of every object.  
yolov4.weights: It is the learned parameters of the Neural Network Model.

yolov4.cfg: It is the file that storage the configuration that specifies the architecture of YoloV4 Neural Network. It is basically the DNN settings.

To load the YoloV4 DNN, must type this command: net = cv2.dnn.readNet(config\_path\_directory, weights\_path\_directory)

When we will pass an image to the Yolo DNN (Deep Neural Network) using OpenCV, it is necessary convert the conventional image format to the blob format. The ***blob format*** is a pre-processed image format for Neural Networks. This processing can resize the image, change the pixel scaling factor, swap from RGB to BGR and other options.

Observation: It is ALWAYS necessary convert the traditional image variable to the blob format to send it to the Neural Network Input!!

blob = cv2.dnn.blobFromImage(image, pixelScalingFactor, (imgWidth, imgHeight), swapRB = True/False, crop = True/False)

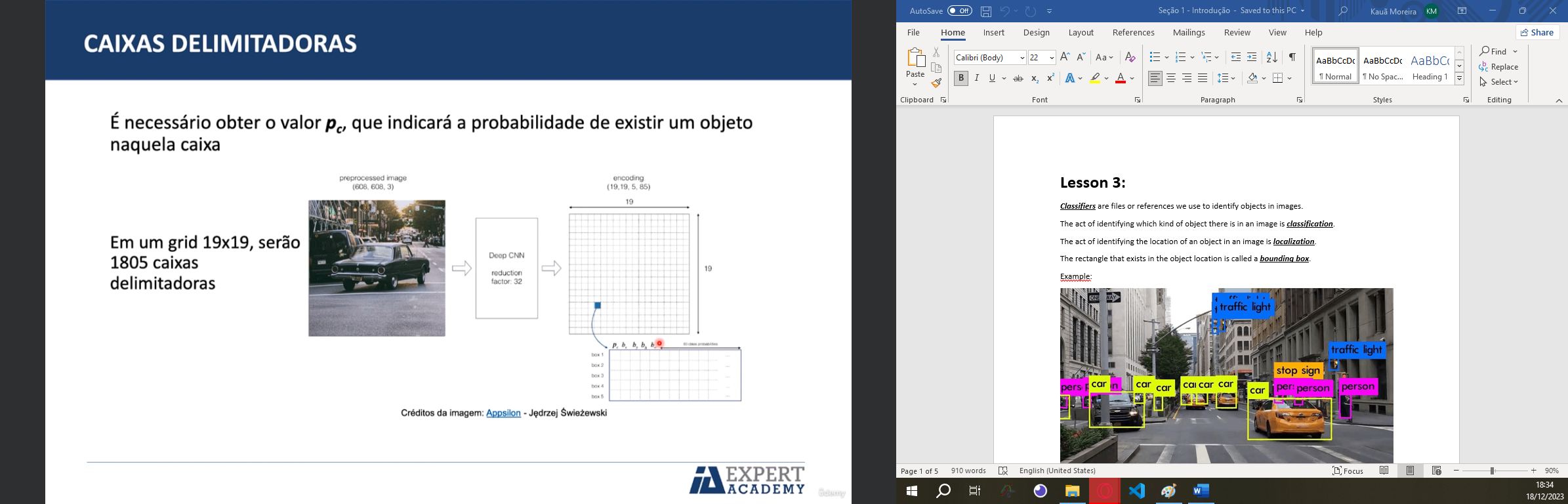
* pixelScalingFactor => It changes the pixel scaling factor. So, 1 / 255.0 is essentially dividing each pixel value by 255. This operation scales down the original pixel values to a range between 0 and 1. For instance, if a pixel initially had a value of 127 (in the range 0-255), applying 1 / 255.0 would result in a normalized value of approximately 0.498 (127 divided by 255).
* (imgWidth, imgHeight) => Define the image width and height.
* swapRB => Define if the image must be converted to the BGR format, because OpenCV just understand BGR.
* Crop => Defines if the image will be cropped after be resized.

After convert to the blob format, it is necessary pass the image to the DNN input and that is the command: net.setInput(blob)

After that, you can simply write the command to the DNN process the image:  
layer\_outputs = net.forward(ln)

It is necessary pass the “ln” parameter to the function. The ln parameter is the name of the output layers. It is necessary pass this information to the function because the function must know which layers expect the output information.

What Yolo give us in its output layers?



Here it is an example to understand the image above:

inicio = time.time()

blob = cv2.dnn.blobFromImage(imagem, 1 / 255.0, (416, 416), swapRB = True, crop = False)# Convert to the blob format

net.setInput(blob)#Put the image in blob format in the DNN input

layer\_outputs = net.forward(ln)#Get the result of the three output layers

termino = time.time()

print('YOLO levou {:.2f} segundos'.format(termino - inicio))

len(layer\_outputs[2][0])

***layer\_outputs***: This likely represents the output of the YOLO model. It's a multi-dimensional array or tensor that contains information about detected objects.

* [2]: Accessing the third element of layer\_outputs. This might correspond to a specific layer or part of the model's output.
* [0]: Accessing the first element within that third element. This might correspond to a specific detection or entry in that layer's output.

The explanation about having 85 values instead of 80 indicates the structure of each detection's information:

Initially, YOLO is trained to detect 80 classes of objects.

However, when examining the length of one specific line or detection within the output, it's observed that there are 85 values present.

Each individual line or detection in the output contains 80 values representing the probabilities or confidences of the object belonging to each of the 80 classes.

Additionally, the remaining 5 values contain information about the bounding box associated with that detection:

* Pc: Probability of an object's existence within that bounding box (regardless of the object's class).
* Bx and By: Starting positions of the bounding box.
* Bh and Bw: Height and width of the bounding box.

So, while the YOLO model is trained to recognize 80 classes, each detection's output includes additional information about the presence of an object in general (Pc) and the specifics of the bounding box (position, width, and height).

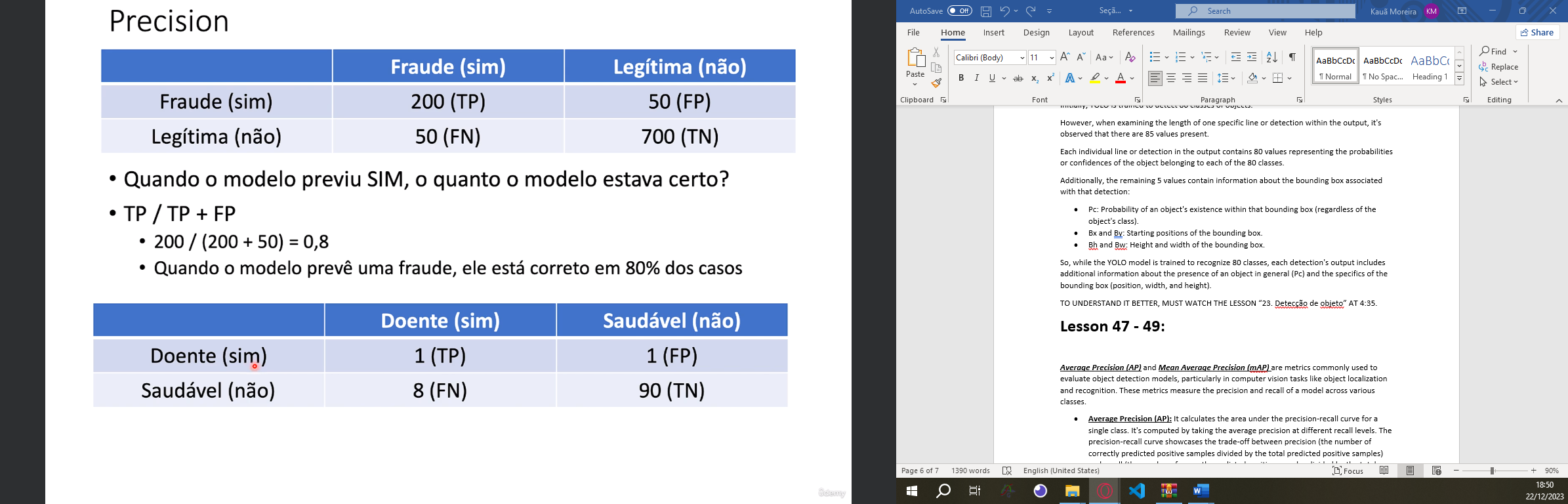
TO UNDERSTAND IT BETTER, MUST WATCH THE LESSON “23. Detecção de objeto” AT 4:35.

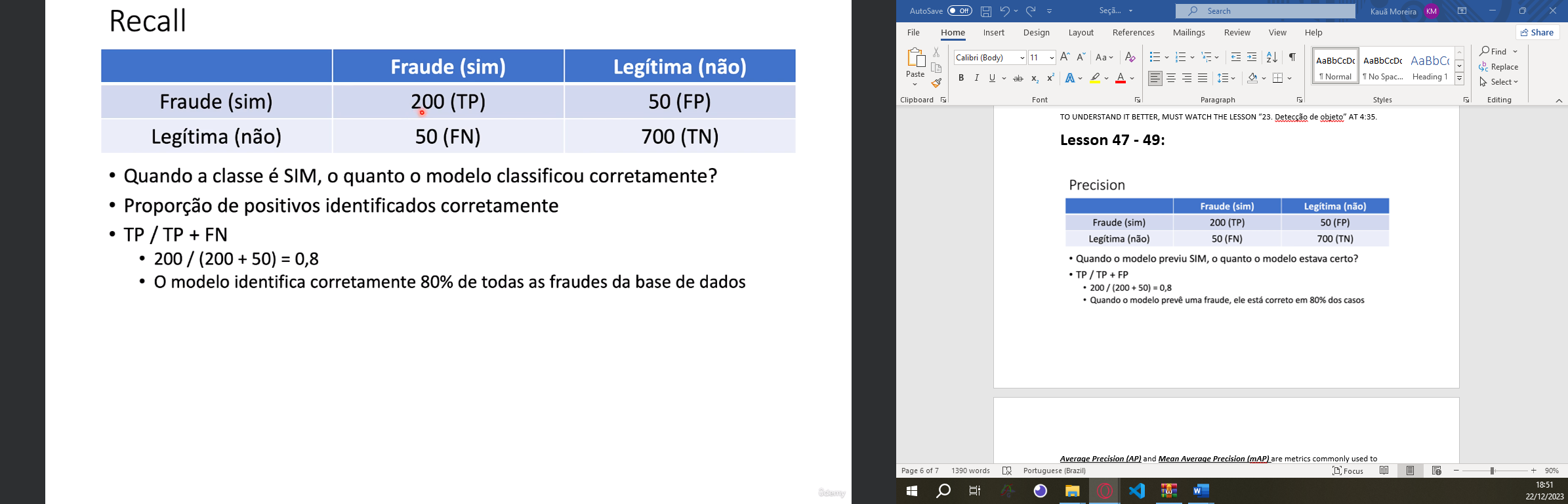
**Lesson 47 - 49:**

What is ***precision*** and ***recall***?

* ***Precision*** is a parameter that is responsible to answer how much porcent the model was right when It predicted “Yes”.
* ***Recall*** is a parameter that is responsible to answer how much porcent the model classified correctly when the class is “Yes”.

Above we have some images as an example:





***Average Precision (AP)*** and ***Mean Average Precision (mAP)*** are metrics commonly used to evaluate object detection models, particularly in computer vision tasks like object localization and recognition. These metrics measure the precision and recall of a model across various classes.

* **Average Precision (AP):** It calculates the area under the precision-recall curve for a single class. It's computed by taking the average precision at different recall levels. The precision-recall curve showcases the trade-off between precision (the number of correctly predicted positive samples divided by the total predicted positive samples) and recall (the number of correctly predicted positive samples divided by the total actual positive samples).
* **Mean Average Precision (mAP):** It's the mean value of AP across multiple classes. For a model that predicts multiple classes, mAP is often used as an aggregate metric to evaluate its overall performance. Each class has its AP calculated, and then the mean of these AP values gives the mAP.

***Intersection over Union (IoU)*** is a metric used to evaluate the accuracy of an object detection algorithm, particularly in scenarios where bounding boxes are predicted around objects within images or videos. It's calculated as the ratio between the area of overlap between the predicted and ground truth bounding boxes and the area of their union:

