Project 4 -Convolutional Neural Network

Due Monday by 11:59pm **Points** 0

This the source files for this project: **P4_ML.zip** (https://canvas.ou.edu/courses/227140/files/53600943/download?download_frd=1)

This project has two parts and each one has their own source code.First you train a Neural Network for Digit Classification. Next you will use Yolo package to detect cars for self-driving cars application:

Delivery: In the first section, you are modifying models.nn in folder Q1_DigitClass and in the section 2,you have to modify car_detect_yolo.py in folder Q2_yolo

Important: Section 2 code is based on Tensorflow, make sure you install all the dependencies.

Section 1: Building Neural Nets

Throughout the first part of the project, you'll use the framework provided in `nn.py` . A simple neural network has layers, where each layer performs a linear operation (just like perceptron). Layers are separated by a *non-linearity*, which allows the network to approximate general functions. We'll use the ReLU operation for our non-linearity, defined as relu(x)=max(x,0)

For example, a simple two-layer neural network for mapping an input row vector x to an output vector f(x) would be given by the function: $f(x)=relu(x\cdot W_1+b_1)\cdot W_2+b_2$ where we have parameter matrices W_1 and W_2 and parameter vectors b_1 and b_2 to learn during gradient descent. W_1 will be an i×h matrix, where i is the dimension of our input vectors x, and h is the *hidden layer size*. b_1 will be a size h vector. We are free to choose any value we want for the hidden size (we will just need to make sure the dimensions of the other matrices and vectors agree so that we can perform the operations). Using a larger hidden size will usually make the network more powerful (able to fit more training data), but can make the network harder to train (since it adds more parameters to all the matrices and vectors we need to learn), or can lead to overfitting on the training data.

We can also create deeper networks by adding more layers, for example a three-layer net:

$f(x)=relu(relu(x\cdot W_1+b_1)\cdot W_2+b_2)\cdot W_3+b_3$

- Batching:

For efficiency, you will be required to process whole batches of data at once rather than a single example at a time. This means that instead of a single input row vector x

with size i, you will be presented with a batch of b inputs represented as a b×i matrix X

. We provide an example for linear regression to demonstrate how a linear layer can be implemented in the batched setting.

Randomness and Initialization:

The parameters of your neural network will be randomly initialized, and data in some tasks will be presented in shuffled order. Due to this randomness, it's possible that you will still occasionally fail some tasks even with a strong architecture – this is the problem of local optima! This should happen very rarely, though – if when testing your code you fail the autograder twice in a row for a question, you should explore other architectures.

- Practical tips

Designing neural nets can take some trial and error. Here are some tips to help you along the way:

- Be systematic. Keep a log of every architecture you've tried, what the hyperparameters (layer sizes, learning rate, etc.) were, and what the resulting performance was. As you try more things, you can start seeing patterns about which parameters matter. If you find a bug in your code, be sure to cross out past results that are invalid due to the bug.
- Start with a shallow network (just two layers, i.e. one non-linearity). Deeper networks have exponentially more hyperparameter combinations, and getting even a single one wrong can ruin your performance. Use the small network to find a good learning rate and layer size; afterwards you can consider adding more layers of similar size.
- If your learning rate is wrong, none of your other hyperparameter choices matter. You can take a state-of-the-art model from a research paper, and change the learning rate such that it performs no better than random. A learning rate too low will result in the model learning too slowly, and a learning rate too high may cause loss to diverge to infinity. Begin by trying different learning rates while looking at how the loss decreases over time.
- Smaller batches require lower learning rates. When experimenting with different batch sizes, be aware that the best learning rate may be different depending on the batch size.
- Refrain from making the network too wide (hidden layer sizes too large) If you keep making the network wider accuracy will gradually decline, and computation time will increase quadratically in the layer size – you're likely to give up due to excessive slowness long before the accuracy falls too much. If your code is taking much longer than 4-5 minutes, you should check it for efficiency.
- If your model is returning Infinity or NaN, your learning rate is probably too high for your current architecture.
- Recommended values for your hyperparameters:
    - Hidden layer sizes: between 10 and 400.
    - Batch size: between 1 and the size of the dataset. For Q2 and Q3, we require that total size of the dataset be evenly divisible by the batch size.

- Learning rate: between 0.001 and 1.0.
- Number of hidden layers: between 1 and 3.

Provided codes:

Here is a full list of nodes available in `nn.py`. You will make use of these in the remaining parts of the assignment:

- `nn.Constant` represents a matrix (2D array) of floating point numbers. It is typically used to represent input features or target outputs/labels. Instances of this type will be provided to you by other functions in the API; you will not need to construct them directly.
- `nn.Parameter` represents a trainable parameter of a perceptron or neural network. All parameters must be 2-dimensional.
  - Usage: `nn.Parameter(n, m)` constructs a parameter with shape n×m
-
- `nn.Add` adds matrices element-wise.
  - Usage: `nn.Add(x, y)` accepts two nodes of shape batch_size×num_features

and constructs a node that also has shape batch_size×num_features

  - .
- `nn.AddBias` adds a bias vector to each feature vector.
  - Usage: `nn.AddBias(features, bias)` accepts `features` of shape batch_size×num_features

and `bias` of shape 1×num_features, and constructs a node that has shape batch_size×num_features

  - .
- `nn.Linear` applies a linear transformation (matrix multiplication) to the input.
  - Usage: `nn.Linear(features, weights)` accepts `features` of shape batch_size×num_input_features

and `weights` of shape num_input_features×num_output_features, and constructs a node that has shape batch_size×num_output_features

- `nn.ReLU` applies the element-wise Rectified Linear Unit nonlinearity relu(x)=max(x,0)
- . This nonlinearity replaces all negative entries in its input with zeros.
  - Usage: `nn.ReLU(features)`, which returns a node with the same shape as the `input`.
- `nn.SquareLoss` computes a batched square loss, used for regression problems
  - Usage: `nn.SquareLoss(a, b)`, where `a` and `b` both have shape batch_size×num_outputs
- `nn.SoftmaxLoss` computes a batched softmax loss, used for classification problems.
  - Usage: `nn.SoftmaxLoss(logits, labels)`, where `logits` and `labels` both have shape batch_size×num_classes
  - . The term “logits” refers to scores produced by a model, where each entry can be an arbitrary real number. The labels, however, must be non-negative and have each row sum to 1. Be sure not to swap the order of the arguments!

- *Do not use* `nn.DotProduct` *for any model other than the perceptron.*

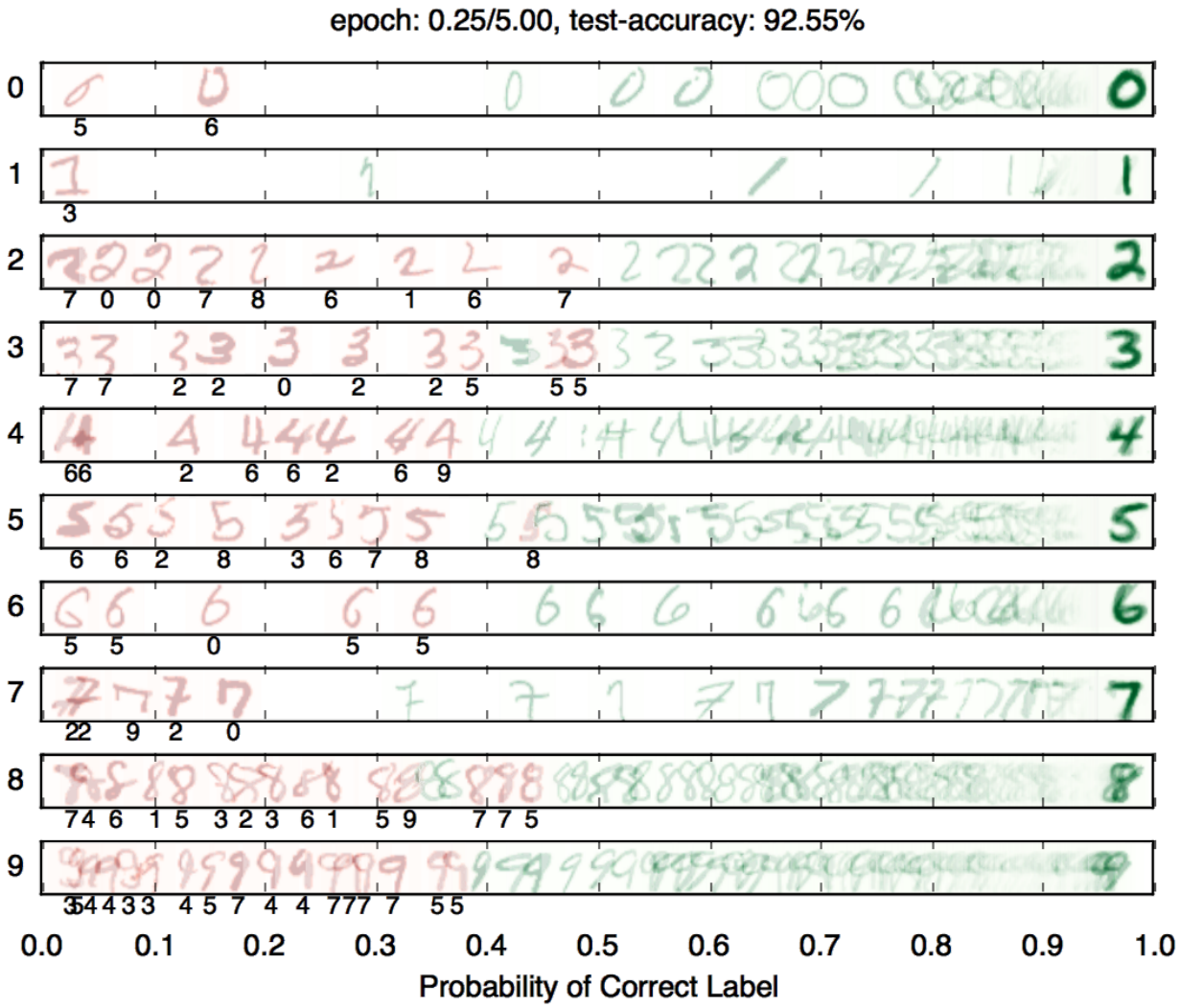
The following methods are available in `nn.py`:

- `nn.gradients` computes gradients of a loss with respect to provided parameters.
  - Usage: `nn.gradients(loss, [parameter_1, parameter_2, ..., parameter_n])` will return a list `[gradient_1, gradient_2, ..., gradient_n]`, where each element is an `nn.Constant` containing the gradient of the loss with respect to a parameter.
- `nn.as_scalar` can extract a Python floating-point number from a loss node. This can be useful to determine when to stop training.
  - Usage: `nn.as_scalar(node)`, where `node` is either a loss node or has shape `(1,1)`.

The datasets provided also have two additional methods:

- `dataset.iterate_forever(batch_size)` yields an infinite sequences of batches of examples.
- `dataset.get_validation_accuracy()` returns the accuracy of your model on the validation set. This can be useful to determine when to stop training

Question 1.1: Digit Classification



For this question, you will train a network to classify handwritten digits from the MNIST dataset.

Each digit is of size 28×28 pixels, the values of which are stored in a 784-dimensional vector of floating point numbers. Each output we provide is a 10-dimensional vector which has zeros in all positions, except for a one in the position corresponding to the correct class of the digit.

Complete the implementation of the `DigitClassificationModel` class in `models.py`. The return value from `DigitClassificationModel.run()` should be a batch_size×10

node containing scores, where higher scores indicate a higher probability of a digit belonging to a particular class (0-9). You should use `nn.SoftmaxLoss` as your loss. Do not put a ReLU activation after the last layer of the network.

Your tasks are to:

- Implement `DigitClassificationModel.__init__` with any needed initialization

- Implement `DigitClassificationModel.run` to return a batch_size×1 node that represents your model's prediction.
- Implement `DigitClassificationModel.get_loss` to return a loss for given inputs and target outputs.
- Implement `DigitClassificationModel.train`, which should train your model using gradient-based updates.

In addition to training data, there is also validation data and a test set. You can use `dataset.get_validation_accuracy()` to compute validation accuracy for your model, which can be useful when deciding whether to stop training. The test set will be used by the autograder.

To receive points for this question, your model should achieve an accuracy of at least 97% on the test set. For reference, our staff implementation consistently achieves an accuracy of 98% on the validation data after training for around 5 epochs. Note that the test grades you on **test accuracy**, while you only have access to **validation accuracy** - so if your validation accuracy meets the 97% threshold, you may still fail the test if your test accuracy does not meet the threshold. Therefore, it may help to set a slightly higher stopping threshold on validation accuracy, such as 97.5% or 98%.

To test your implementation, run the autograder:

```
python autograder.py -q q3
```

Section 2: Autonomous Driving - Car Detection using YOLO!

(https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autono Driving---Car-Detection)

In this section, you'll implement object detection using the very powerful YOLO model. Many of the ideas in this assignment are described in the two YOLO papers: **Redmon et al., 2016 (https://arxiv.org/abs/1506.02640)** and **Redmon and Farhadi, 2016 (https://arxiv.org/abs/1612.08242)**.

This assignment is elected from Depp Learning course in Coursera **here (https://www.coursera.org/learn/convolutional-neural-networks/programming/3VCFG/car-detection-with-yolo/lab?path=%2Fnotebooks%2F.ipynb)**. However, you may follow-up all the steps in this page as well

By the end of this assignment, you'll be able to:

- Detect objects in a car detection dataset
- Implement non-max suppression to increase accuracy
- Implement intersection over union
- Handle bounding boxes, a type of image annotation popular in deep learning

Packages

The source code can be found here as archive. For this part of the project, you need to modify car_detect_yolo.py.

In this project we write the code stepby step. The file car_detect_yolo.py initially includes the following lines. To make sure all required packages are installed, run the original file:

```
python car_detect_yolo.py
```

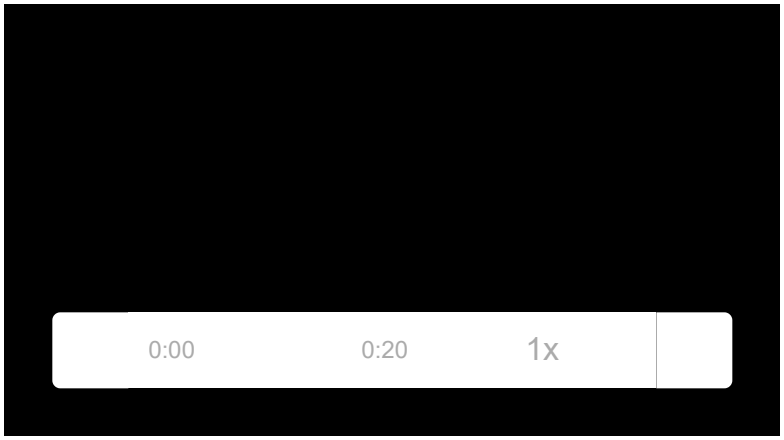
```
import argparse

import os

import matplotlib.pyplot as plt

from matplotlib.pyplot import imshow

import scipy.io

import scipy.misc

import numpy as np

import pandas as pd

import PIL
from Keras import backend as K

from PIL import ImageFont, ImageDraw, Image

import tensorflow as tf

from tensorflow.python.framework.ops import EagerTensor

from yolo_utils import read_classes, read_anchors,scale_boxes, preprocess_image

from tensorflow.keras.models import load_model

from yad2k.models.keras_yolo import yolo_head

from yad2k.utils.draw_boxes import get_colors_for_classes
```

2.1 - Problem Statement

(https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_driving_application_Car_d

--Problem-Statement)

You are working on a self-driving car. Go you! As a critical component of this project, you'd like to first build a car detection system. To collect data, you've mounted a camera to the hood (meaning the front) of the car, which takes pictures of the road ahead every few seconds as you drive around.



Pictures taken from a car-mounted camera while driving around Silicon Valley.
Dataset provided by **drive.ai** **(https://www.drive.ai/)**.

You've gathered all these images into a folder and labelled them by drawing bounding boxes around every car you found. Here's an example of what your bounding boxes look like:

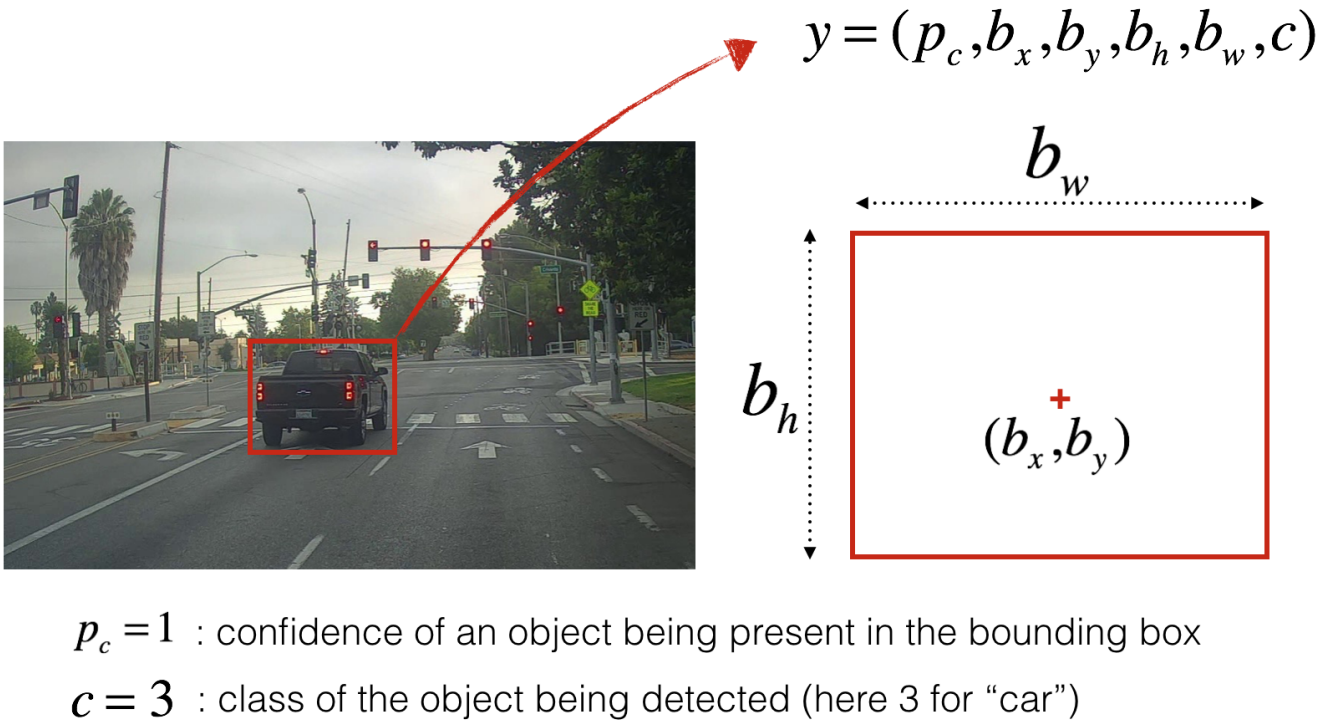


Figure 1: Definition of a box

If there are 80 classes you want the object detector to recognize, you can represent the class label either as an integer from 1 to 80, or as an 80-dimensional vector (with 80 numbers) one component of which is 1, and the rest of which are 0.

In this exercise, you'll discover how YOLO ("You Only Look Once") performs object detection, and then apply it to car detection. Because the YOLO model is very computationally expensive to train, the pre-trained weights are already loaded for you to use.

2.2 - YOLO

(https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autono--YOLO)

"You Only Look Once" (YOLO) is a popular algorithm because it achieves high accuracy while also being able to run in real time. This algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the network to make predictions. After non-max suppression, it then outputs recognized objects together with the bounding boxes.

2.2.1 - Model Details

Inputs and outputs

(https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_driving_application_Car_dete and-outputs)

- The **input** is a batch of images, and each image has the shape (m, 608, 608, 3)
- The **output** is a list of bounding boxes along with the recognized classes. Each bounding box is represented by 6 numbers (p_c, b_x,b_y,b_h,b_w,c)

as explained above. If you expand

- into an 80-dimensional vector, each bounding box is then represented by 85 numbers.

Anchor Boxes

(https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_driving_application_Car_dete Boxes)

- Anchor boxes are chosen by exploring the training data to choose reasonable height/width ratios that represent the different classes. For this assignment, 5 anchor boxes were chosen for you (to cover the 80 classes), and stored in the file './model_data/yolo_anchors.txt'
- The dimension of the encoding tensor of the second to last dimension based on the anchor boxes is (m, n_H,n_W, anchors, classes)
- The YOLO architecture is: IMAGE (m, 608, 608, 3) -> DEEP CNN -> ENCODING (m, 19, 19, 5, 85).

Encoding

(https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_drivir

Let's look in greater detail at what this encoding represents.

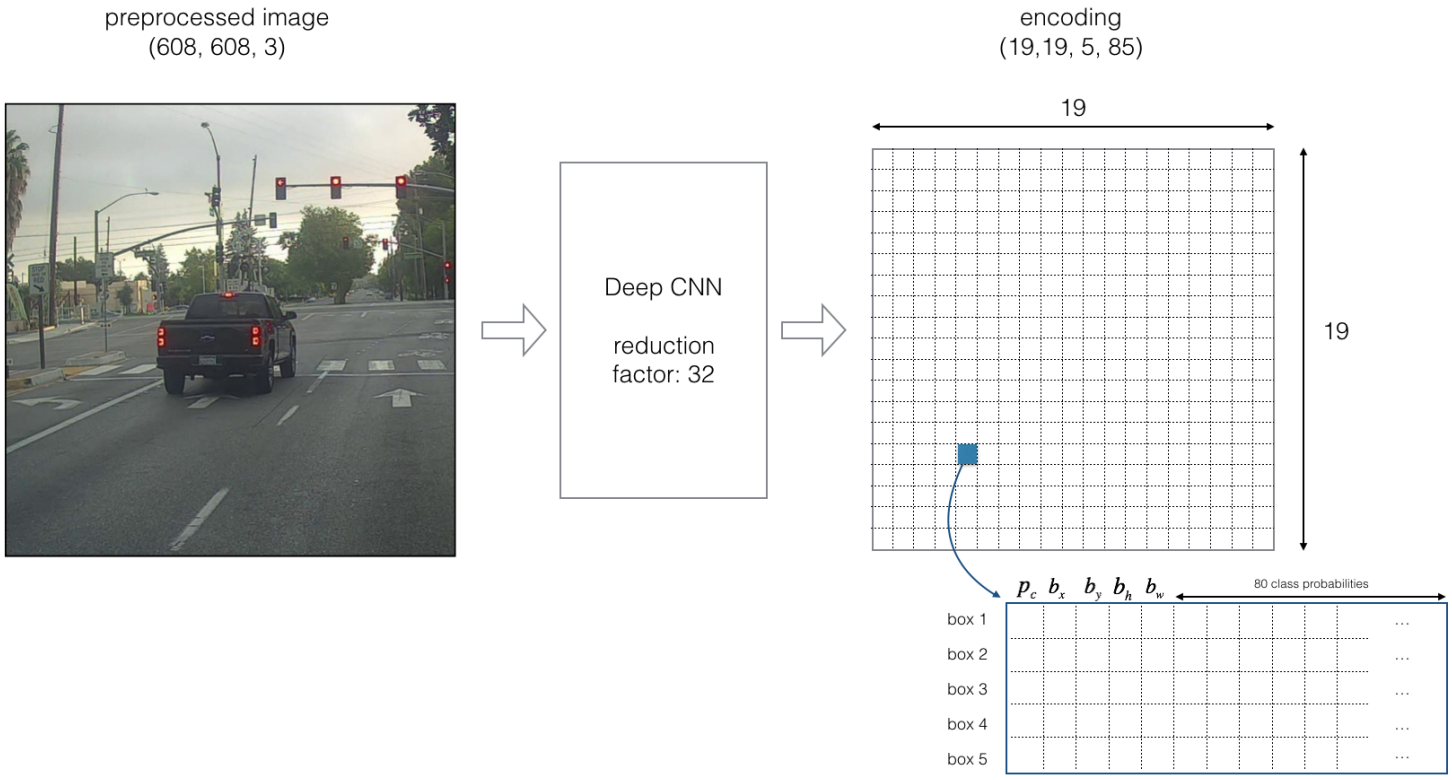


Figure 2 : Encoding architecture for YOLO

If the center/midpoint of an object falls into a grid cell, that grid cell is responsible for detecting that object.

Since you're using 5 anchor boxes, each of the 19 x19 cells thus encodes information about 5 boxes. Anchor boxes are defined only by their width and height.

For simplicity, you'll flatten the last two dimensions of the shape (19, 19, 5, 85) encoding, so the output of the Deep CNN is (19, 19, 425).

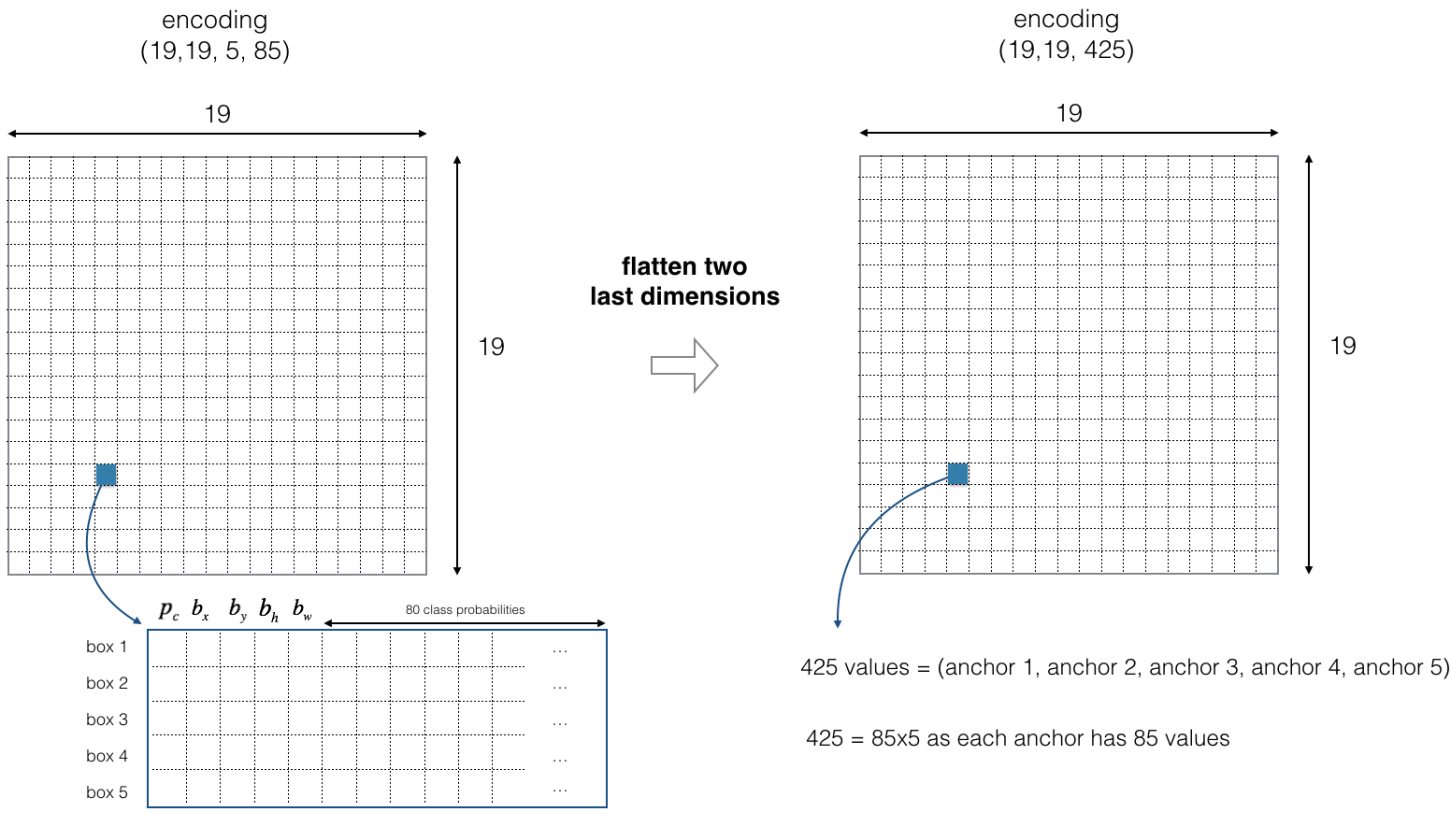


Figure 3 : Flattening the last two last dimensions

Class score

(https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_drivir score)

Now, for each box (of each cell) you'll compute the following element-wise product and extract a probability that the box contains a certain class.
The class score is $score_{c,i} = p_c \times c_i$, the probability that there is an object p_c times the probability that the object is a certain class c_i.

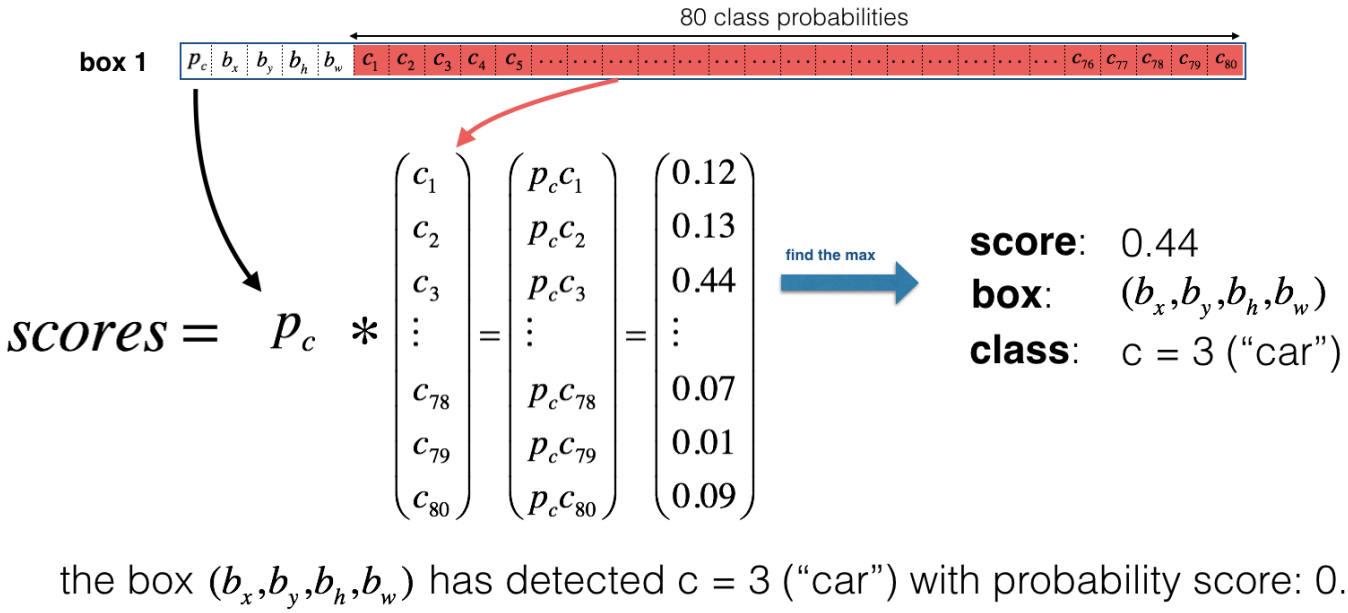


Figure 4: find the class detected by each box

Example of figure 4 (https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_drivir of-figure-4)

- In figure 4, let's say for box 1 (cell 1), the probability that an object exists is
- . So there's a 60% chance that an object exists in box 1 (cell 1).
- The probability that the object is the class "category 3 (a car)" is $c_3 = 0.73$
- The score for box 1 and for category "3" is : $score_{1,3} = 0.6 \times 0.73 = 0.44$

- Let's say you calculate the score for all 80 classes in box 1, and find that the score for the car class (class 3) is the maximum. So you'll assign the score 0.44 and class "3" to this box "1".

Visualizing classes (https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_drivir classes)

Here's one way to visualize what YOLO is predicting on an image:

- For each of the 19x19 grid cells, find the maximum of the probability scores (taking a max across the 80 classes, one maximum for each of the 5 anchor boxes).
- Color that grid cell according to what object that grid cell considers the most likely.

Doing this results in this picture:

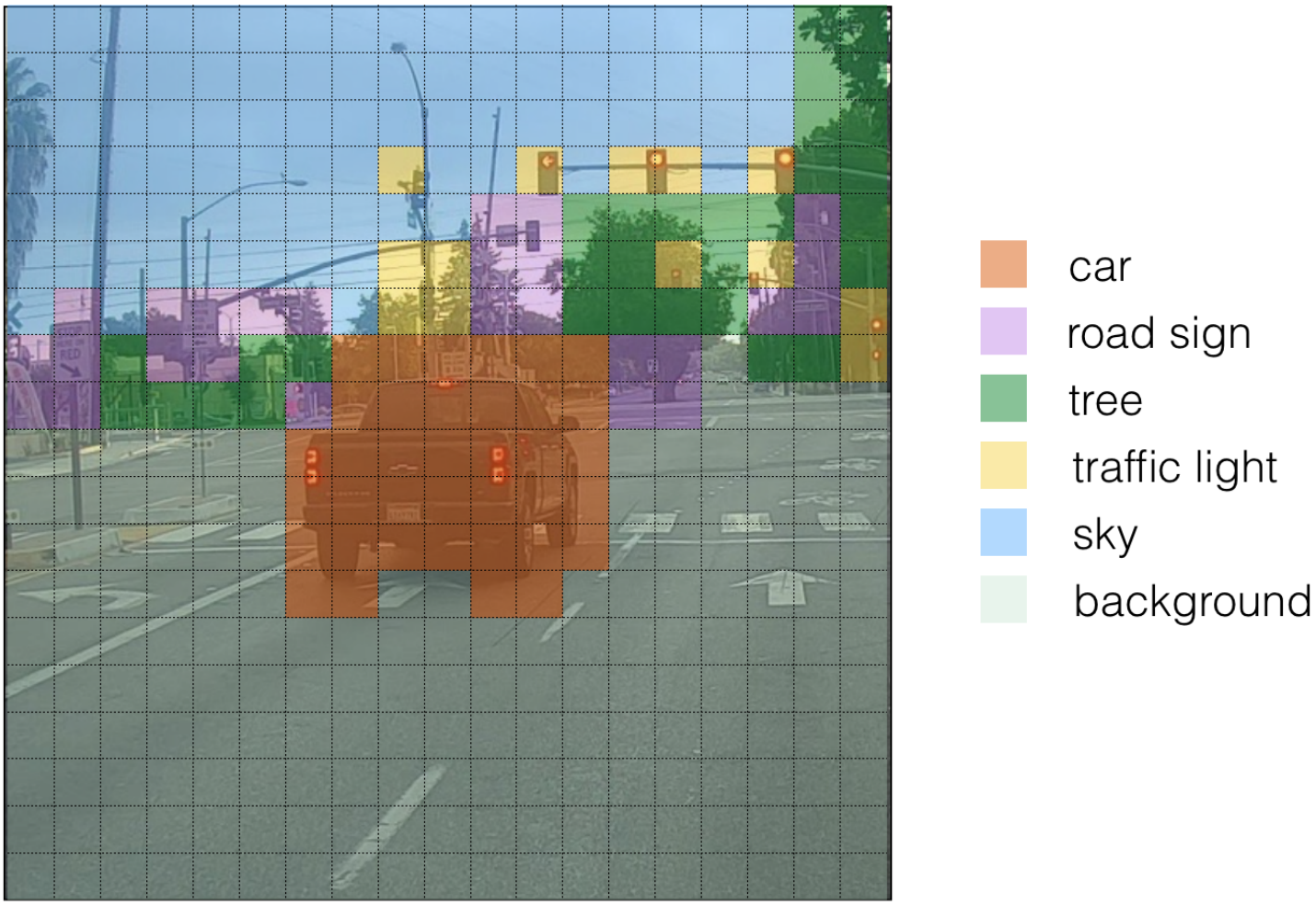


Figure 5: Each one of the 19x19 grid cells is colored according to which class has the largest predicted probability in that cell.

Note that this visualization isn't a core part of the YOLO algorithm itself for making predictions; it's just a nice way of visualizing an intermediate result of the algorithm.

Visualizing bounding boxes

(https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_drivir bounding-boxes)

Another way to visualize YOLO's output is to plot the bounding boxes that it outputs. Doing that results in a visualization like this:

Figure 6: Each cell gives you 5 boxes. In total, the model predicts: 19x19x5 = 1805 boxes just by looking once at the image (one forward pass through the network)! Different colors denote different classes.

Non-Max suppression

(https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_drivir Max-suppression)

In the figure above, the only boxes plotted are ones for which the model had assigned a high probability, but this is still too many boxes. You'd like to reduce the algorithm's output to a much smaller number of detected objects.

To do so, you'll use **non-max suppression**. Specifically, you'll carry out these steps:

- Get rid of boxes with a low score. Meaning, the box is not very confident about detecting a class, either due to the low probability of any object, or low probability of this particular class.
- Select only one box when several boxes overlap with each other and detect the same object.

2.2.2 - Filtering with a Threshold on Class Scores (https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_drivir--Filtering-with-a-Threshold-on-Class-Scores)

You're going to first apply a filter by thresholding, meaning you'll get rid of any box for which the class "score" is less than a chosen threshold.

The model gives you a total of 19x19x5x85 numbers, with each box described by 85 numbers. It's convenient to rearrange the (19,19,5,85) (or (19,19,425)) dimensional tensor into the following variables:

- `box_confidence`: tensor of shape $(19, 19, 5, 1)$ containing p_c (confidence probability that there's some object) for each of the 5 boxes predicted in each of the 19x19 cell
- b`oxes`: tensor of shape `(19,19,5,1)containing the midpoint and dimensions for` (b_x, b_y, b_h, b_w)
- `box_class_probs`: tensor of shape$(19, 19, 5, 80)$ containing the "class probabilities" $(c_1, c_2, \ldots, c_{80})$ for each of the 80 classes for each of the 5 boxes per cell.

Question 2.1: yolo_filter_boxes (https://cfqwtxip.labs.coursera.org/notebooks/W3A1/Autonomous_drivir1---yolo_filter_boxes)

Implement `yolo_filter_boxes()`.

1. Compute box scores by doing the element -wise product as described in Figure 4 ($p \times c$)
2. The following code may help you choose the right operator:

```
a = np.random.randn(19, 19, 5, 1)
b = np.random.randn(19, 19, 5, 80)
c = a * b # shape of c will be (19, 19, 5, 80)
```

This is an example of **broadcasting** (multiplying vectors of different sizes).

For each box, find:

- the index of the class with the maximum box score
- the corresponding box score

Useful References

- tf.math.argmax (https://www.tensorflow.org/api_docs/python/tf/math/argmax)
- tf.math.reduce_max (https://www.tensorflow.org/api_docs/python/tf/math/reduce_max)

Helpful Hints

- For the `axis` parameter of `argmax` and `reduce_max`, if you want to select the **last** axis, one way to do so is to set `axis=-1`. This is similar to Python array indexing, where you can select the last position of an array using `arrayname[-1]`.
- Applying `reduce_max` normally collapses the axis for which the maximum is applied. `keepdims=False` is the default option, and allows that dimension to be removed. You don't need to keep the last dimension after applying the maximum here.

3. Create a mask by using a threshold. As a reminder: `([0.9, 0.3, 0.4, 0.5, 0.1] < 0.4)` returns: `[False, True, False, False, True]`. The mask should be `True` for the boxes you want to keep.

4. Use TensorFlow to apply the mask to `box_class_scores`, `boxes` and `box_classes` to filter out the boxes you don't want. You should be left with just the subset of boxes you want to keep.

   One more useful reference:

   - **tf.boolean mask** (https://www.tensorflow.org/api_docs/python/tf/boolean_mask)

   And one more helpful hint:

   - For the `tf.boolean_mask`, you can keep the default `axis=None`.

Delivery: you have to add the following function and complete it:

```
# UNQ_C1 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: yolo_filter_boxes

def yolo_filter_boxes(boxes, box_confidence, box_class_probs, threshold = .6):
    """Filters YOLO boxes by thresholding on object and class confidence.

    Arguments:
        boxes -- tensor of shape (19, 19, 5, 4)
        box_confidence -- tensor of shape (19, 19, 5, 1)
        box_class_probs -- tensor of shape (19, 19, 5, 80)
        threshold -- real value, if [ highest class probability score < threshold],
                     then get rid of the corresponding box

    Returns:
        scores -- tensor of shape (None,), containing the class probability score for selected boxes
        boxes -- tensor of shape (None, 4), containing (b_x, b_y, b_h, b_w) coordinates of selected boxes
        classes -- tensor of shape (None,), containing the index of the class detected by the selected boxes

    Note: "None" is here because you don't know the exact number of selected boxes, as it depends on the thresh
    For example, the actual output size of scores would be (10,) if there are 10 boxes.
    """

    ### START CODE HERE
    # Step 1: Compute box scores
    ##(≈ 1 line)
    box_scores = None

    # Step 2: Find the box_classes using the max box_scores, keep track of the corresponding score
    ##(≈ 2 lines)
    # IMPORTANT: set axis to -1
```

```
    box_classes = None
    box_class_scores = None

    # Step 3: Create a filtering mask based on "box_class_scores" by using "threshold". The mask should have th
    # same dimension as box_class_scores, and be True for the boxes you want to keep (with probability >= thres
    ## (≈ 1 line)
    filtering_mask = None

    # Step 4: Apply the mask to box_class_scores, boxes and box_classes
    ## (≈ 3 lines)
    scores = None
    boxes = None
    classes = None
    ### END CODE HERE

    return scores, boxes, class
```

Unit test:

Run the following unit test as auto grader

```
# BEGIN UNIT TEST
tf.random.set_seed(10)
box_confidence = tf.random.normal([19, 19, 5, 1], mean=1, stddev=4, seed = 1)
boxes = tf.random.normal([19, 19, 5, 4], mean=1, stddev=4, seed = 1)
box_class_probs = tf.random.normal([19, 19, 5, 80], mean=1, stddev=4, seed = 1)
scores, boxes, classes = yolo_filter_boxes(boxes, box_confidence, box_class_probs, threshold = 0.5)
print("scores[2] = " + str(scores[2].numpy()))
print("boxes[2] = " + str(boxes[2].numpy()))
print("classes[2] = " + str(classes[2].numpy()))
print("scores.shape = " + str(scores.shape))
print("boxes.shape = " + str(boxes.shape))
print("classes.shape = " + str(classes.shape))

assert type(scores) == EagerTensor, "Use tensorflow functions"
assert type(boxes) == EagerTensor, "Use tensorflow functions"
assert type(classes) == EagerTensor, "Use tensorflow functions"

assert scores.shape == (1789,), "Wrong shape in scores"
assert boxes.shape == (1789, 4), "Wrong shape in boxes"
assert classes.shape == (1789,), "Wrong shape in classes"

assert np.isclose(scores[2].numpy(), 9.270486), "Values are wrong on scores"
assert np.allclose(boxes[2].numpy(), [4.6399336, 3.2303846, 4.431282, -2.202031]), "Values are wrong on boxes"
assert classes[2].numpy() == 8, "Values are wrong on classes"

print("\033[92m All tests passed!")
# END UNIT TEST
```

Expected Output:

scores[2]	9.270486
boxes[2]	[4.6399336 3.2303846 4.431282 -2.202031]
classes[2]	8

scores.shape (1789,)

boxes.shape (1789, 4)

classes.shape (1789,)

Note In the test for `yolo_filter_boxes`, you're using random numbers to test the function. In real data, the non-zero values between 0 and 1 for the probabilities. The box coordinates in `boxes` would also be chose non-negative.

2.3 - Non-max Suppression (https://cfqwtxip.labs.coursera.org/notebo max-Suppression)

Even after filtering by thresholding over the class scores, you still end up with a lot of overlapping boxes. right boxes is called non-maximum suppression (NMS).

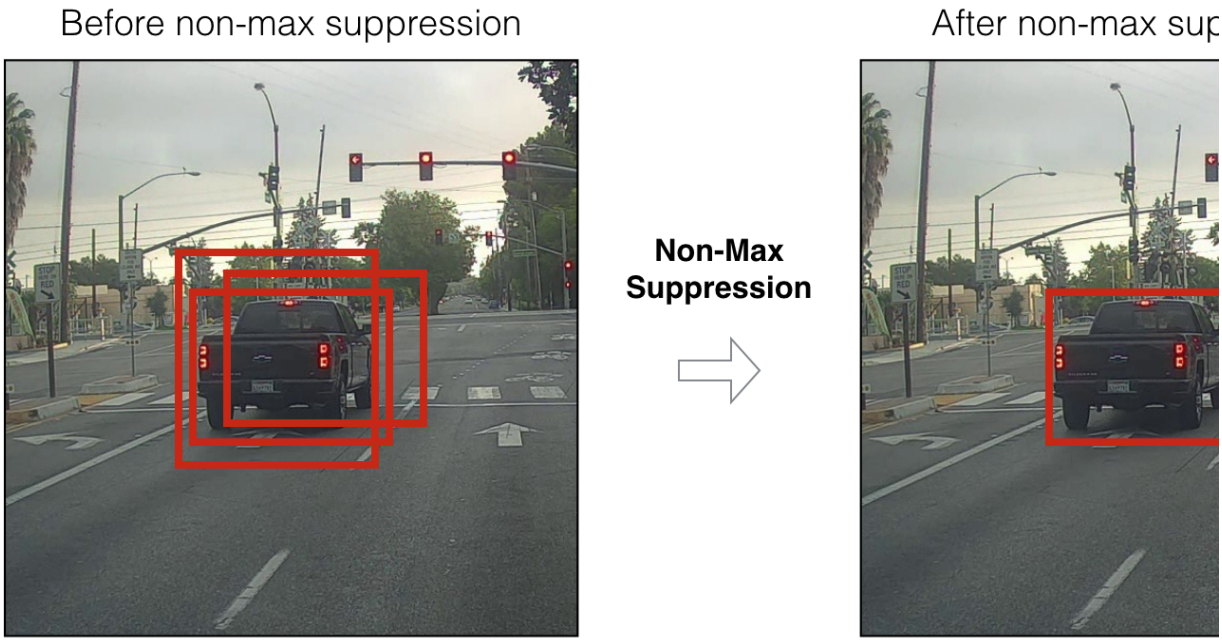


Figure 7 : In this example, the model has predicted 3 cars, but it's actually 3 predictions of the same car. (NMS) will select only the most accurate (highest probability) of the 3 boxes.

Non-max suppression uses the very important function called **"Intersection over Union"**, or IoU.

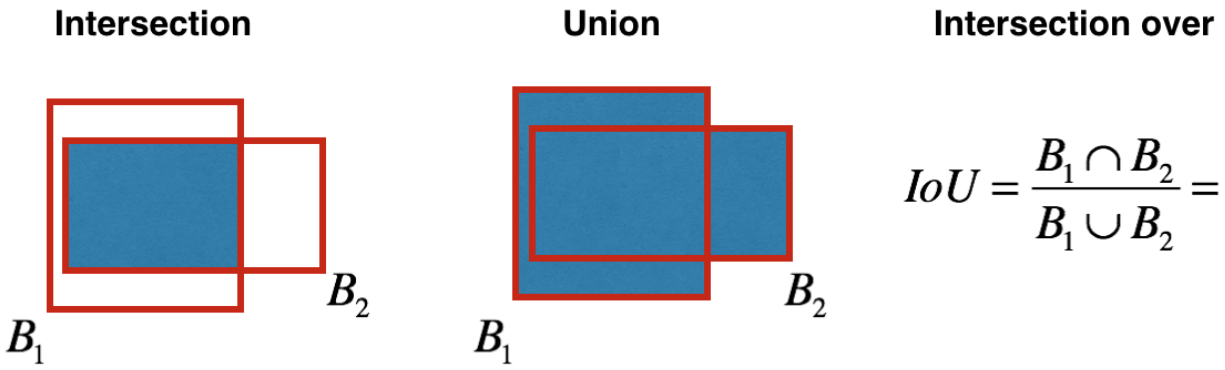


Figure 8 : Definition of "Intersection over Union".

Question 2.2 - iou (https://cfqwtxip.labs.coursera.org/notebooks/.ipynb

In this question, you will implement function `iou()`

Some hints:

- This code uses the convention that (0,0) is the top-left corner of an image, (1,0) is the upper-right corr corner. In other words, the (0,0) origin starts at the top left corner of the image. As x increases, you m you move down.
- For this exercise, a box is defined using its two corners: upper left (x_1, y_1) and lower right (x_2, y_2)
- , instead of using the midpoint, height and width. This makes it a bit easier to calculate the intersectio
- To calculate the area of a rectangle, multiply its height $x_2 - x_1$ by its width $y_2 - y_1$. Since $(x1, y1)$ are the bottom right, these differences should be non-negative.
- To find the **intersection** of the two boxes $(xi1, yi1, xi2, yi2)$
- Feel free to draw some examples on paper to clarify this conceptually
- The top left corner of the intersection (xi1,yi1) is found by comparing the top left corners (x1,y1) of the that has an x-coordinate that is closer to the right, and y-coordinate that is closer to the bottom.
- The bottom right corner of the intersection (xi2,yi2) is found by comparing the bottom right corners (finding a vertex whose x-coordinate is closer to the left, and the y-coordinate that is closer to the top.
- The two boxes **may have no intersection**. You can detect this if the intersection coordinates you calc and/or bottom left corners of an intersection box. Another way to think of this is if you calculate the he find that at least one of these lengths is negative, then there is no intersection (intersection area is zer
- The two boxes may intersect at the **edges or vertices**, in which case the intersection area is still zerc height or width (or both) of the calculated intersection is zero.

Additional Hints

- `xi1` = **max**imum of the x1 coordinates of the two boxes
- `yi1` = **max**imum of the y1 coordinates of the two boxes
- `xi2` = **min**imum of the x2 coordinates of the two boxes
- `yi2` = **min**imum of the y2 coordinates of the two boxes
- `inter_area` = You can use `max(height, 0)` and `max(width, 0)`

```
# UNQ_C2 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: iou

def iou(box1, box2):
    """Implement the intersection over union (IoU) between box1 and box2

    Arguments:
    box1 -- first box, list object with coordinates (box1_x1, box1_y1, box1_x2, box_1_y2)
    box2 -- second box, list object with coordinates (box2_x1, box2_y1, box2_x2, box2_y2)
    """
    (box1_x1, box1_y1, box1_x2, box1_y2) = box1
    (box2_x1, box2_y1, box2_x2, box2_y2) = box2

    ### START CODE HERE
    # Calculate the (yi1, xi1, yi2, xi2) coordinates of the intersection of box1 and box2. Calculate its Area.
    ##(≈ 7 lines)
    xi1 = None
    yi1 = None
    xi2 = None
    yi2 = None
    inter_width = None
    inter_height =  None
    inter_area = None

    # Calculate the Union area by using Formula: Union(A,B) = A + B - Inter(A,B)
    ## (≈ 3 lines)
    box1_area = None
    box2_area = None
    union_area = None

    # compute the IoU
    iou = None
    ### END CODE HERE

    return iou
```

Unit test:

Run the following unit test as auto grade

```
# BEGIN UNIT TEST
## Test case 1: boxes intersect
box1 = (2, 1, 4, 3)
box2 = (1, 2, 3, 4)

print("iou for intersecting boxes = " + str(iou(box1, box2)))
assert iou(box1, box2) < 1, "The intersection area must be always smaller or equal than the union area."
assert np.isclose(iou(box1, box2), 0.14285714), "Wrong value. Check your implementation. Problem with intersect

## Test case 2: boxes do not intersect
box1 = (1,2,3,4)
box2 = (5,6,7,8)
print("iou for non-intersecting boxes = " + str(iou(box1,box2)))
assert iou(box1, box2) == 0, "Intersection must be 0"

## Test case 3: boxes intersect at vertices only
box1 = (1,1,2,2)
box2 = (2,2,3,3)
```

```
print("iou for boxes that only touch at vertices = " + str(iou(box1,box2)))
assert iou(box1, box2) == 0, "Intersection at vertices must be 0"

## Test case 4: boxes intersect at edge only
box1 = (1,1,3,3)
box2 = (2,3,3,4)
print("iou for boxes that only touch at edges = " + str(iou(box1,box2)))
assert iou(box1, box2) == 0, "Intersection at edges must be 0"

print("\033[92m All tests passed!")
# END UNIT TEST
```

Expected Output:

```
iou for intersecting boxes = 0.14285714285714285
iou for non-intersecting boxes = 0.0
iou for boxes that only touch at vertices = 0.0
iou for boxes that only touch at edges = 0.0
```

2.4 YOLO Non-max Suppression

You are now ready to implement non-max suppression. The key steps are:

1. Select the box that has the highest score.
2. Compute the overlap of this box with all other boxes, and remove boxes that overlap significantly (iou
3. Go back to step 1 and iterate until there are no more boxes with a lower score than the currently selec

This will remove all boxes that have a large overlap with the selected boxes. Only the "best" boxes remai

Question 2.3 : yolo_non_max_suppression

Implement `yolo_non_max_suppression()` using TensorFlow. TensorFlow has two built-in functions that are us suppression (so you don't actually need to use your `iou()` implementation):

Reference documentation:

- **tf.image.non_max_suppression() (https://www.tensorflow.org/api_docs/python/tf/image/non_max_s**

  ```
  tf.image.non_max_suppression(
    boxes,
    scores,
    max_output_size,
    iou_threshold=0.5,
    name=None
  )
  ```

  Note that in the version of TensorFlow used here, there is no parameter `score_threshold` (it's shown in version) so trying to set this value will result in an error message: *got an unexpected keyword argume*

- **tf.gather() (https://www.tensorflow.org/api_docs/python/tf/gather)**

  ```
  keras.gather(
    reference,
  ```

```
    indices
)
```

```
# UNQ_C3 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: yolo_non_max_suppression

def yolo_non_max_suppression(scores, boxes, classes, max_boxes = 10, iou_threshold = 0.5):
    """
    Applies Non-max suppression (NMS) to set of boxes

    Arguments:
    scores -- tensor of shape (None,), output of yolo_filter_boxes()
    boxes -- tensor of shape (None, 4), output of yolo_filter_boxes() that have been scaled to the image size (
    classes -- tensor of shape (None,), output of yolo_filter_boxes()
    max_boxes -- integer, maximum number of predicted boxes you'd like
    iou_threshold -- real value, "intersection over union" threshold used for NMS filtering

    Returns:
    scores -- tensor of shape (None, ), predicted score for each box
    boxes -- tensor of shape (None, 4), predicted box coordinates
    classes -- tensor of shape (None, ), predicted class for each box

    Note: The "None" dimension of the output tensors has obviously to be less than max_boxes. Note also that th
    function will transpose the shapes of scores, boxes, classes. This is made for convenience.
    """

    max_boxes_tensor = tf.Variable(max_boxes, dtype='int32')     # tensor to be used in tf.image.non_max_suppre

    ### START CODE HERE
    # Use tf.image.non_max_suppression() to get the list of indices corresponding to boxes you keep
    ##(≈ 1 line)
    nms_indices = None

    # Use tf.gather() to select only nms_indices from scores, boxes and classes
    ##(≈ 3 lines)
    scores = None
    boxes = None
    classes = None
    ### END CODE HERE

    return scores, boxes, classes
```

And for testing the code, use the following unit test:

```
# BEGIN UNIT TEST
tf.random.set_seed(10)
scores = tf.random.normal([54,], mean=1, stddev=4, seed = 1)
boxes = tf.random.normal([54, 4], mean=1, stddev=4, seed = 1)
classes = tf.random.normal([54,], mean=1, stddev=4, seed = 1)
scores, boxes, classes = yolo_non_max_suppression(scores, boxes, classes)

assert type(scores) == EagerTensor, "Use tensoflow functions"
print("scores[2] = " + str(scores[2].numpy()))
print("boxes[2] = " + str(boxes[2].numpy()))
print("classes[2] = " + str(classes[2].numpy()))
print("scores.shape = " + str(scores.numpy().shape))
print("boxes.shape = " + str(boxes.numpy().shape))
print("classes.shape = " + str(classes.numpy().shape))

assert type(scores) == EagerTensor, "Use tensoflow functions"
assert type(boxes) == EagerTensor, "Use tensoflow functions"
assert type(classes) == EagerTensor, "Use tensoflow functions"
```

```
assert scores.shape == (10,), "Wrong shape"
assert boxes.shape == (10, 4), "Wrong shape"
assert classes.shape == (10,), "Wrong shape"

assert np.isclose(scores[2].numpy(), 8.147684), "Wrong value on scores"
assert np.allclose(boxes[2].numpy(), [ 6.0797963, 3.743308, 1.3914018, -0.34089637]), "Wrong value on boxes"
assert np.isclose(classes[2].numpy(), 1.7079165), "Wrong value on classes"

print("\033[92m All tests passed!")
# END UNIT TEST
```

Expected Output:

scores[2]	8.147684
boxes[2]	[6.0797963 3.743308 1.3914018 -0.34089637]
classes[2]	1.7079165
scores.shape	(10,)
boxes.shape	(10, 4)
classes.shape	(10,)

2.5 Wrapping Up the Filtering

It's time to implement a function taking the output of the deep CNN (the 19x19x5x85 dimensional encodir boxes using the functions you've just implemented.

Question 2.4 - yolo_eval

Implement `yolo_eval()` which takes the output of the YOLO encoding and filters the boxes using score th one last implementational detail you have to know. There're a few ways of representing boxes, such as vi midpoint and height/width. YOLO converts between a few such formats at different times, using the follow provided):

```
boxes = yolo_boxes_to_corners(box_xy, box_wh)
```

which converts the yolo box coordinates (x,y,w,h) to box corners' coordinates (x1, y1, x2, y2) to fit the inp

```
boxes = scale_boxes(boxes, image_shape)
```

YOLO's network was trained to run on 608x608 images. If you are testing this data on a different size ima detection dataset had 720x1280 images -- this step rescales the boxes so that they can be plotted on top

Don't worry about these two functions; you'll see where they need to be called below.

```
def yolo_boxes_to_corners(box_xy, box_wh):
    """Convert YOLO box predictions to bounding box corners."""
    box_mins = box_xy - (box_wh / 2.)
    box_maxes = box_xy + (box_wh / 2.)
```

```
    return tf.keras.backend.concatenate([
        box_mins[..., 1:2],  # y_min
        box_mins[..., 0:1],  # x_min
        box_maxes[..., 1:2],  # y_max
        box_maxes[..., 0:1]  # x_max
    ])
```

```
# UNQ_C4 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: yolo_eval

def yolo_eval(yolo_outputs, image_shape = (720, 1280), max_boxes=10, score_threshold=.6, iou_threshold=.5):
    """
    Converts the output of YOLO encoding (a lot of boxes) to your predicted boxes along with their scores, box

    Arguments:
    yolo_outputs -- output of the encoding model (for image_shape of (608, 608, 3)), contains 4 tensors:
                    box_xy: tensor of shape (None, 19, 19, 5, 2)
                    box_wh: tensor of shape (None, 19, 19, 5, 2)
                    box_confidence: tensor of shape (None, 19, 19, 5, 1)
                    box_class_probs: tensor of shape (None, 19, 19, 5, 80)
    image_shape -- tensor of shape (2,) containing the input shape, in this notebook we use (608., 608.) (has t
    max_boxes -- integer, maximum number of predicted boxes you'd like
    score_threshold -- real value, if [ highest class probability score < threshold], then get rid of the corre
    iou_threshold -- real value, "intersection over union" threshold used for NMS filtering

    Returns:
    scores -- tensor of shape (None, ), predicted score for each box
    boxes -- tensor of shape (None, 4), predicted box coordinates
    classes -- tensor of shape (None,), predicted class for each box
    """

    ### START CODE HERE
    # Retrieve outputs of the YOLO model (≈1 line)
    box_xy, box_wh, box_confidence, box_class_probs = None

    # Convert boxes to be ready for filtering functions (convert boxes box_xy and box_wh to corner coordinates)
    boxes = None

    # Use one of the functions you've implemented to perform Score-filtering with a threshold of score_threshol
    scores, boxes, classes = None

    # Scale boxes back to original image shape.
    boxes = None

    # Use one of the functions you've implemented to perform Non-max suppression with
    # maximum number of boxes set to max_boxes and a threshold of iou_threshold (≈1 line)
    scores, boxes, classes = None
    ### END CODE HERE

    return scores, boxes, classes
```

Run the unit test for this section:

```
# BEGIN UNIT TEST
tf.random.set_seed(10)
yolo_outputs = (tf.random.normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1),
                tf.random.normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1),
                tf.random.normal([19, 19, 5, 1], mean=1, stddev=4, seed = 1),
                tf.random.normal([19, 19, 5, 80], mean=1, stddev=4, seed = 1))
scores, boxes, classes = yolo_eval(yolo_outputs)
print("scores[2] = " + str(scores[2].numpy()))
print("boxes[2] = " + str(boxes[2].numpy()))
print("classes[2] = " + str(classes[2].numpy()))
print("scores.shape = " + str(scores.numpy().shape))
```

```
print("boxes.shape = " + str(boxes.numpy().shape))
print("classes.shape = " + str(classes.numpy().shape))

assert type(scores) == EagerTensor, "Use tensoflow functions"
assert type(boxes) == EagerTensor, "Use tensoflow functions"
assert type(classes) == EagerTensor, "Use tensoflow functions"

assert scores.shape == (10,), "Wrong shape"
assert boxes.shape == (10, 4), "Wrong shape"
assert classes.shape == (10,), "Wrong shape"

assert np.isclose(scores[2].numpy(), 171.60194), "Wrong value on scores"
assert np.allclose(boxes[2].numpy(), [-1240.3483, -3212.5881, -645.78, 2024.3052]), "Wrong value on boxes"
assert np.isclose(classes[2].numpy(), 16), "Wrong value on classes"

print("\033[92m All tests passed!")
# END UNIT TEST
```

Expected Output:

scores[2]	171.60194
boxes[2]	[-1240.3483 -3212.5881 -645.78 2024.3052]
classes[2]	16
scores.shape	(10,)
boxes.shape	(10, 4)
classes.shape	(10,)

2.6 - Test YOLO Pre-trained Model on Images (https://cfqwtxip.labs.coursera.org/notebooks/.ipynb#3---Test-trained-Model-on-Images)

In this section, you are going to use a pre-trained model and test it on the car detection dataset.

2.6.1 - Defining Classes, Anchors and Image Shape (https://cfqwtxip.labs.coursera.org/notebooks/.ipynb#3.1---Defining-Class-Image-Shape)

You're trying to detect 80 classes, and are using 5 anchor boxes. The information on the 80 classes and 5 "coco_classes.txt" and "yolo_anchors.txt". You'll read class names and anchors from text files. The car de images, which are pre-processed into 608x608 images.

```
class_names = read_classes("model_data/coco_classes.txt")
anchors = read_anchors("model_data/yolo_anchors.txt")
```

```
model_image_size = (608, 608) # Same as yolo_model input layer size
```

2.6.2 - Loading a Pre-trained Model

Training a YOLO model takes a very long time and requires a fairly large dataset of labelled bounding bo
classes. You are going to load an existing pre-trained Keras YOLO model stored in "yolo.h5". These weig
website, and were converted using a function written by Allan Zelener. References are at the end of this r
the parameters from the "YOLOv2" model, but are simply referred to as "YOLO" in this notebook.

Run the cell below to load the model from this file.

```
yolo_model = load_model("model_data/", compile=False)
```

This loads the weights of a trained YOLO model. Here's a summary of the layers your model contains:

```
yolo_model.summary()
```

Note: On some computers, you may see a warning message from Keras. Don't worry about it if you do --

Reminder: This model converts a preprocessed batch of input images (shape: (m, 608, 608, 3)) into a te
explained in Figure (2).

2.6.3 - Convert Output of the Model to Usable Bounding Box Tensors (https://cfqwtxip.labs.coursera.org/notebooks/.ipynb#3.3---Convert-Outp Usable-Bounding-Box-Tensors)

The output of `yolo_model` is a (m, 19, 19, 5, 85) tensor that needs to pass through non-trivial processing a
call `yolo_head` to format the encoding of the model you got from `yolo_model` into something decipherable:

yolo_model_outputs = yolo_model(image_data) yolo_outputs = yolo_head(yolo_model_outputs, anchors
`yolo_outputs` will be defined as a set of 4 tensors that you can then use as input by your yolo_eval functio
yolo_head is implemented, you can find the function definition in the file `keras_yolo.py`. The file is also loc
path: `yad2k/models/keras_yolo.py`.

2.6.4 - Filtering Boxes (https://cfqwtxip.labs.coursera.org/notebooks/.i Boxes)

`yolo_outputs` gave you all the predicted boxes of `yolo_model` in the correct format. To perform filtering and
will call `yolo_eval`, which you had previously implemented, to do so:

```
out_scores, out_boxes, out_classes = yolo_eval(yolo_outputs, [image.size[1],  image.size[0]], 10, 0.3, 0.5)
```

Question 2.5 - Run the YOLO on an Image

Let the fun begin! You will create a graph that can be summarized as follows:

`yolo_model.input` is given to `yolo_model`. The model is used to compute the output `yolo_model.output` `yolo_` `yolo_head`. It gives you `yolo_outputs` `yolo_outputs` goes through a filtering function, `yolo_eval`. It outputs y `out_boxes`, `out_classes`.

Now, we have implemented for you the `predict(image_file)` function, which runs the graph to test YOLO c `out_scores`, `out_boxes`, `out_classes`.

The code below also uses the following function:

```
image, image_data = preprocess_image("images/" + image_file, model_image_size = (608, 608))
```

which opens the image file and scales, reshapes and normalizes the image. It returns the outputs:

```
image: a python (PIL) representation of your image used for drawing boxes. You won't need to use it.
image_data: a numpy-array representing the image. This will be the input to the CNN.
```

```
def predict(image_file):
    """
    Runs the graph to predict boxes for "image_file". Prints and plots the predictions.

    Arguments:
    image_file -- name of an image stored in the "images" folder.

    Returns:
    out_scores -- tensor of shape (None, ), scores of the predicted boxes
    out_boxes -- tensor of shape (None, 4), coordinates of the predicted boxes
    out_classes -- tensor of shape (None, ), class index of the predicted boxes

    Note: "None" actually represents the number of predicted boxes, it varies between 0 and max_boxes.
    """

    # Preprocess your image
    image, image_data = preprocess_image("images/" + image_file, model_image_size = (608, 608))

    yolo_model_outputs = yolo_model(image_data)
    yolo_outputs = yolo_head(yolo_model_outputs, anchors, len(class_names))

    out_scores, out_boxes, out_classes = yolo_eval(yolo_outputs, [image.size[1],  image.size[0]], 10, 0.3, 0.5)

    # Print predictions info
    print('Found {} boxes for {}'.format(len(out_boxes), "images/" + image_file))
    # Generate colors for drawing bounding boxes.
    colors = get_colors_for_classes(len(class_names))
    # Draw bounding boxes on the image file
    #draw_boxes2(image, out_scores, out_boxes, out_classes, class_names, colors, image_shape)
    draw_boxes(image, out_boxes, out_classes, class_names, out_scores)
    # Save the predicted bounding box on the image
    image.save(os.path.join("out", image_file), quality=100)
    # Display the results in the notebook
    output_image = Image.open(os.path.join("out", image_file))
    imshow(output_image)

    return out_scores, out_boxes, out_classes
```

Run the following cell on the "test.jpg" image to verify that your function is correct.

```
out_scores, out_boxes, out_classes = predict("test.jpg")
```

Expected Output:

Found 10 boxes for images/test.jpg	
car	0.89 (367, 300) (745, 648)
car	0.80 (761, 282) (942, 412)
car	0.74 (159, 303) (346, 440)
car	0.70 (947, 324) (1280, 705)
bus	0.67 (5, 266) (220, 407)
car	0.66 (706, 279) (786, 350)
car	0.60 (925, 285) (1045, 374)
car	0.44 (336, 296) (378, 335)
car	0.37 (965, 273) (1022, 292)
traffic light	00.36 (681, 195) (692, 214)

The model you've just run is actually able to detect 80 different classes listed in "coco_classes.txt". To tes

If you were to run your session in a for loop over all your images. Here's what you would get:

Predictions of the YOLO model on pictures taken from a camera while driving around the Silicon Valley
Thanks to **drive.ai** **(https://www.drive.ai/)** for providing this dataset!

4 - Summary for YOLO

(https://cfqwtxip.labs.coursera.org/notebooks/.ipynb#4---Su

- Input image (608, 608, 3)
- The input image goes through a CNN, resulting in a (19,19,5,85) dimensional output.
- After flattening the last two dimensions, the output is a volume of shape (19, 19, 425):
  - Each cell in a 19x19 grid over the input image gives 425 numbers.
  - 425 = 5 x 85 because each cell contains predictions for 5 boxes, corresponding to 5 anchor boxes
  - 85 = 5 + 80
  - has 5 numbers, and 80 is the number of classes we'd like to detect
- You then select only few boxes based on:
  - Score-thresholding: throw away boxes that have detected a class with a score less than the thresh
  - Non-max suppression: Compute the Intersection over Union and avoid selecting overlapping boxe
- This gives you YOLO's final output.

What you should remember:

- YOLO is a state-of-the-art object detection model that is fast and accurate
- It runs an input image through a CNN, which outputs a 19x19x5x85 dimensional volume.
- The encoding can be seen as a grid where each of the 19x19 cells contains information about 5 boxe
- You filter through all the boxes using non-max suppression. Specifically:
  - Score thresholding on the probability of detecting a class to keep only accurate (high probability) b
  - Intersection over Union (IoU) thresholding to eliminate overlapping boxes
- Because training a YOLO model from randomly initialized weights is non-trivial and requires a large d computation, previously trained model parameters were used in this exercise. If you wish, you can als with your own dataset, though this would be a fairly non-trivial exercise.

Congratulations! You've come to the end of this assignment.

Here's a quick recap of all you've accomplished.

You've:

- Detected objects in a car detection dataset
- Implemented non-max suppression to achieve better accuracy
- Implemented intersection over union as a function of NMS
- Created usable bounding box tensors from the model's predictions

Amazing work! If you'd like to know more about the origins of these ideas, spend some time on the paper

5 - References (https://cfqwtxip.labs.coursera.org/noteboo

References)

The ideas presented in this notebook came primarily from the two YOLO papers. The implementation her and used many components from Allan Zelener's GitHub repository. The pre-trained weights used in this YOLO website.

- Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi - **You Only Look Once: Unified, Real- (https://arxiv.org/abs/1506.02640)** (2015)
- Joseph Redmon, Ali Farhadi - **YOLO9000: Better, Faster, Stronger** **(https://arxiv.org/abs/1612.0824**
- Allan Zelener - **YAD2K: Yet Another Darknet 2 Keras** **(https://github.com/allanzelener/YAD2K)**
- The official YOLO website (**https://pjreddie.com/darknet/yolo/** **(https://pjreddie.com/darknet/yolo/)**

Car detection dataset **(https://cfqwtxip.labs.coursera.org/notebooks/.i dataset)**

 **(http://creativecommons.org/licenses/by/4.0/)**

The Drive.ai Sample Dataset (provided by drive.ai) is licensed under a **Creative Commons Attribution 4 (http://creativecommons.org/licenses/by/4.0/)** . Thanks to Brody Huval, Chih Hu and Rahul Patel for provid