Big Data Course Project

Object Detection with YOLOv3

Lorenzo Levantesi 1785694

YOLOv3 - Introduction

YOLOv3

YOLO (You Only Look Once) is an **object detection** model, devised with the aim of being faster than its competitors (e.g., Faster R-CNN, SSD, etc…).

This is possible given that YOLO is composed by a **single neural network** which predicts:

- Bounding Boxes
- Classes

The advantage of this architecture is that the **optimization** phase can be performed **end-to-end**.

But also for a single forward pass the computation time is significantly lower than the **two-stage object detectors**.

YOLOv3 - Architecture

Each **object detection** model is composed by two components:

1. The backbone
2. The detection layers

YOLOv3 uses as **backbone network** the **Darknet-53**, a convolutional neural network with 53 convolutional layers.

For the **detection layers** YOLOv3 predicts bounding boxes by extracting feature maps from Darknet-53

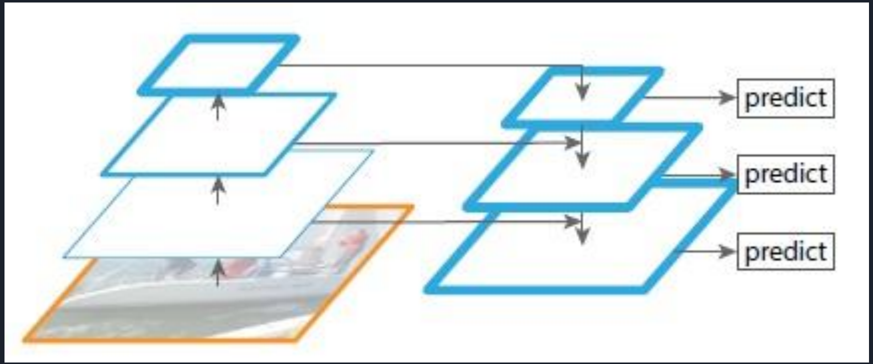
YOLOv3 - Backbone

Darknet-53

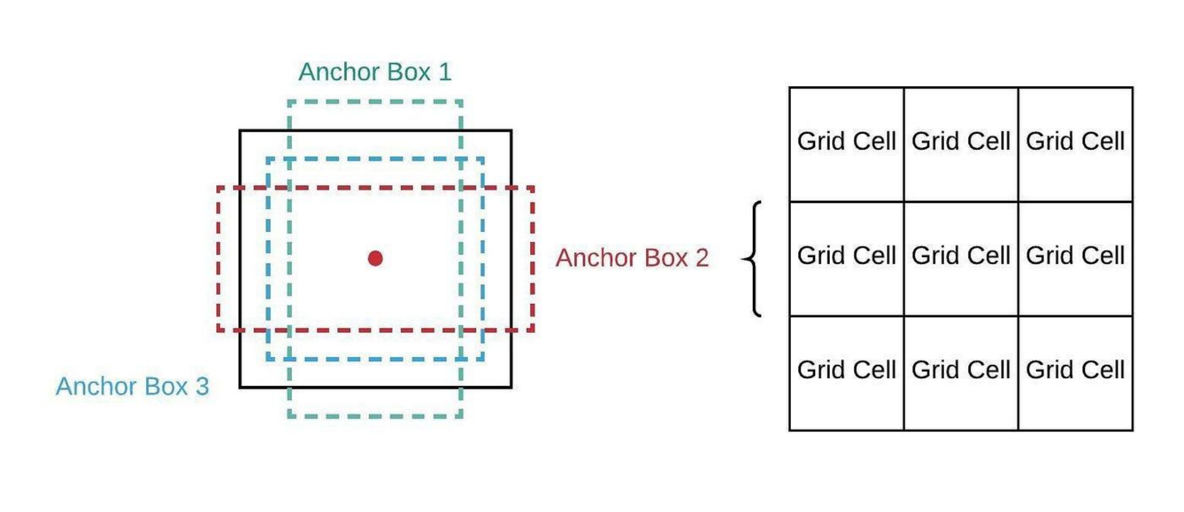
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

YOLOv3 - Detection Layers

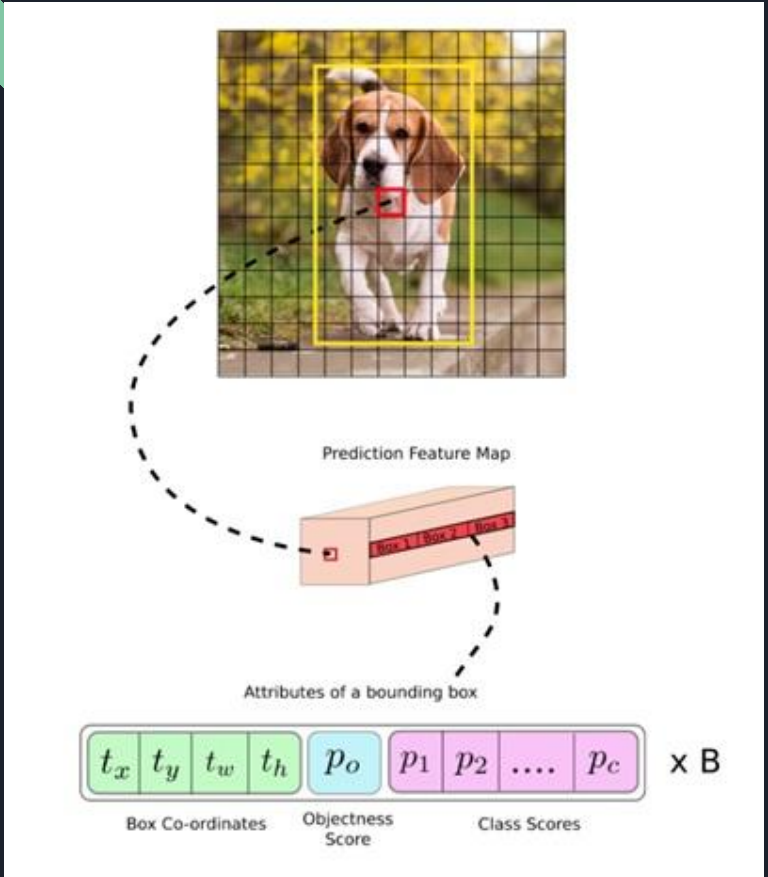
Feature Pyramid Networks

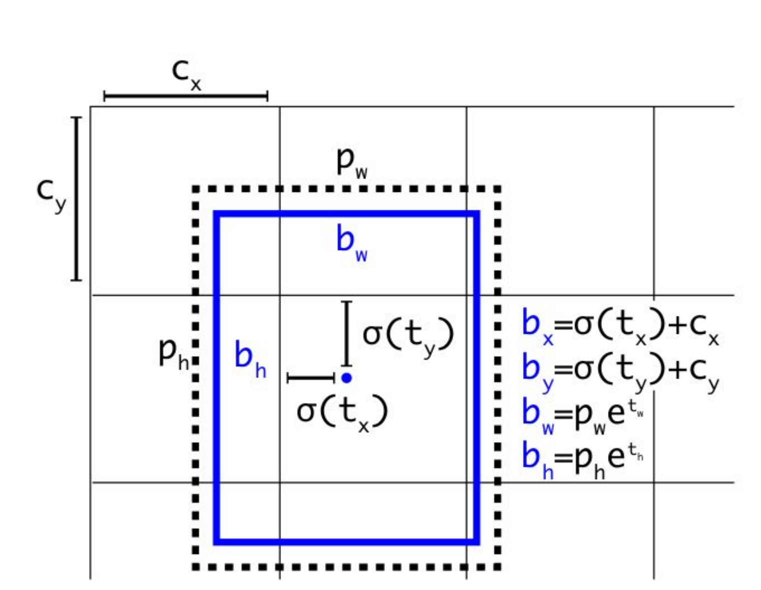


YOLOv3 - Anchor boxes

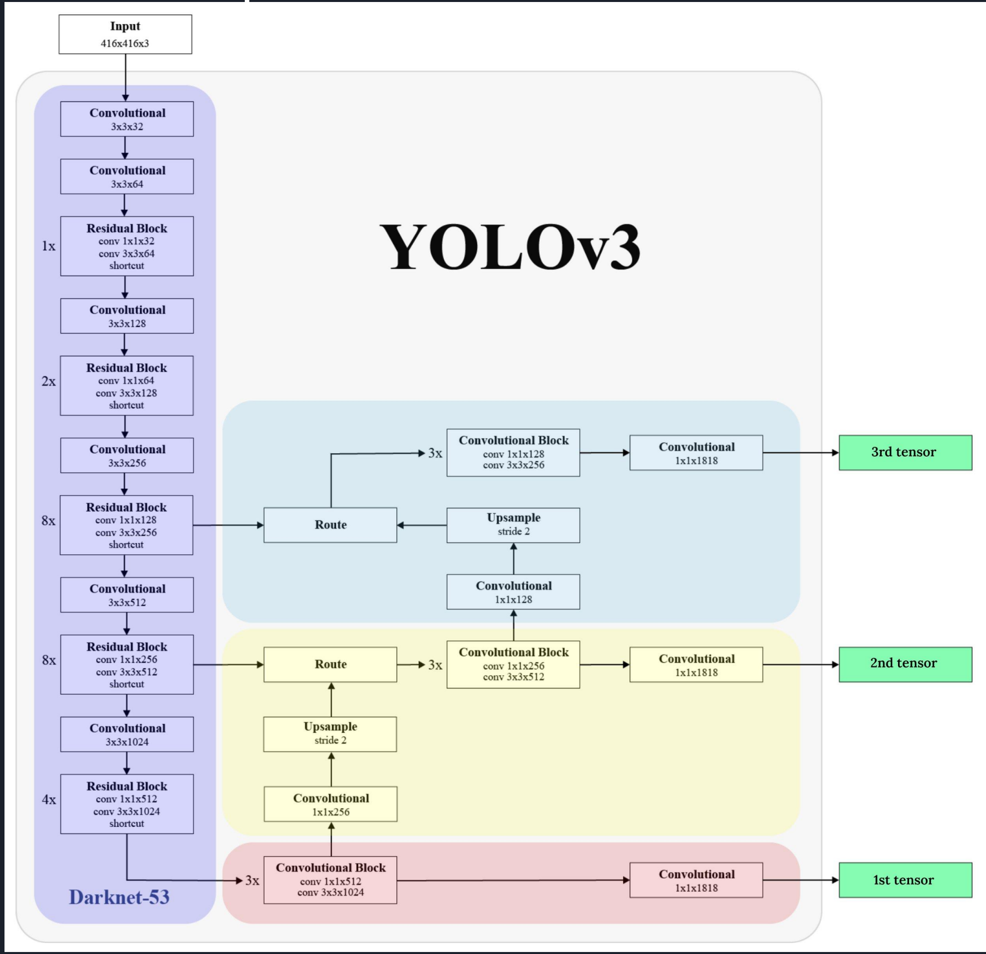


YOLOv3 - Final predictions





YOLOv3 - Complete Network



PyTorch implementation of Darknet-53

Darknet-53 - ImageNet Dataset

The dataset used to train Darknet is the **<u>ImageNet</u>**'s subset ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset for image classification and localization, available on Kaggle.

It contains images for 1,000 object classes, for a total of 1,281,167 training images.

Due to memory limitations, 500 images for each class have been used for training, resulting in a **<u>training set</u>** of 500,000 images.

The **<u>test set</u>** is composed by 100 images for each class, for a total of 100,000 images.

Darknet-53 - Model

```

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
```

Darknet-53 - Model

```
darknet_architecture = [
    (32, 3, 1),
    (64, 3, 2),
    ["residual", 1],
    (128, 3, 2),
    ["residual", 2],
    (256, 3, 2),
    ["residualYolo", 8],
    (512, 3, 2),
    ["residualYolo", 8],
    (1024, 3, 2),
    ["residual", 4],
    ["avgpool"],
    ["prediction"],
    ["softmax"]
]
```

Darknet-53 - Training with SparkTorch

With the aim of simulate a distributed training of the model, I've tried to use **SparkTorch**, a python framework used to train and test PyTorch models in a distributed setting, using APIs from **PySpark**.

But, the first **problem** is that SparkTorch let to train a PyTorch model only on inputs which can be represented in a single column of a PySpark DataFrame, with the constraint of be 1-dimensional.

Given that the images are 3-dimensional, one workaround that I found is:

1. Load the images with the **PySpark's Image data source**, which loads image files from a directory and store them into raw image representation.
2. Modify the **source code of SparkTorch** to transform the bytes image representation into a 3 dimensional Torch tensor.

It worked.

Darknet-53 - Training with SparkTorch

But a new problem arised.

SparkTorch, from what I understood, only works with PyTorch models that have as layers only default torch.nn modules, **without accepting custom torch layers**.

Implementing Darknet-53 with only default layers is really complicated, so I gave up with the idea to train Darknet-53 with SparkTorch.

SparkTorch is a really interesting tool, it has not so much contributions from the community, which makes it a really basic framework for distributed training.

Darkn[illegible]raining with Spark[illegible]

But a new proble[illegible]

SparkTorch, from wha[illegible]vith PyTorch models that have as layers only torch.nn m[illegible]**ting custom torch layers**.

Implementing Darknet-5[illegible]ers is really complicated, so I gave up with the idea to trai[illegible]rch.

SparkTorch is[illegible]esting tool, it has [illegible]ntributions from the community[illegible] it a really basic framew[illegible]ibuted training.

Darknet-53 - Training with SparkTorch

Yesterday I gave it the **last chance**, and I figure it out what was the problem.

In the forward pass, Darknet's PyTorch implementation makes a reference to one of the custom PyTorch subclasses of torch.nn, and this was generating an error.

So, I modified a little bit the code and the distributed training worked.

```
Partition: b50d5df0-70fe-460e-9e54-46dcf1c5e80d. Iteration: 0. Distributed Loss: None Partition Training Loss: 6.171
970367431641, Partition Validation Loss: None
Partition: b50d5df0-70fe-460e-9e54-46dcf1c5e80d. Iteration: 1. Distributed Loss: None Partition Training Loss: 8.947
731971740723, Partition Validation Loss: None
Partition: b50d5df0-70fe-460e-9e54-46dcf1c5e80d. Iteration: 2. Distributed Loss: None Partition Training Loss: 7.474
391937255859, Partition Validation Loss: None
Partition: b50d5df0-70fe-460e-9e54-46dcf1c5e80d. Iteration: 3. Distributed Loss: None Partition Training Loss: 4.196
128845214844, Partition Validation Loss: None
Partition: b50d5df0-70fe-460e-9e54-46dcf1c5e80d. Iteration: 4. Distributed Loss: None Partition Training Loss: 1.999
499797821045, Partition Validation Loss: None
Partition: b50d5df0-70fe-460e-9e54-46dcf1c5e80d. Iteration: 5. Distributed Loss: None Partition Training Loss: 3.570
050001144409, Partition Validation Loss: None
Partition: b50d5df0-70fe-460e-9e54-46dcf1c5e80d. Iteration: 6. Distributed Loss: None Partition Training Loss: 1.652
0180702209473, Partition Validation Loss: None
Partition: b50d5df0-70fe-460e-9e54-46dcf1c5e80d. Iteration: 7. Distributed Loss: None Partition Training Loss: 0.785
6677770614624, Partition Validation Loss: None
Partition: b50d5df0-70fe-460e-9e54-46dcf1c5e80d. Iteration: 8. Distributed Loss: None Partition Training Loss: 12.35
6720924377441, Partition Validation Loss: None
Partition: b50d5df0-70fe-460e-9e54-46dcf1c5e80d. Iteration: 9. Distributed Loss: None Partition Training Loss: 0.894
193708896637, Partition Validation Loss: None
```

Darknet-53 - Training with SparkTorch

```
# load images
df = spark.read.format("image").load("./images")

# create User Defined Function to retrieve labels
labels_udf = F.udf(lambda idx: labels[idx-1])

# Create a column with continuous increasing Id's
df = df.withColumn("num_id", row_number().over(Window.orderBy(monotonically_increasing_id())))

# Create a label column by calling the user defined function
new_df = df.withColumn('label', labels_udf('num_id'))

# Remove Id's column
df = new_df.drop("num_id")
```

Darknet-53 - Training with SparkTorch

```python
# Init model
classes = 1000
darknet = darknet53(darknet_architecture, classes).to(device)

# create Torch Object
torch_obj = sparktorch.serialize_torch_obj(
    model=darknet,
    criterion = nn.CrossEntropyLoss(),
    optimizer = torch.optim.Adam,
    lr=1e-3
)

# create Spark Model
spark_model = sparktorch.SparkTorch(
    inputCol='data',
    labelCol='label',
    predictionCol='predictions',
    torchObj=torch_obj,
    iters=10,
    verbose=1,
    miniBatch=8
)

# execute Spark ML pipeline to start training
p = Pipeline(stages=[spark_model]).fit(df.select(col("image.data"),col("label")))

# retrieve darknet model
darknet = spark_model.getPytorchModel()
```

Darknet-53 - Training

The model trained for almost **5 days**, using the following settings:

- **Optimizer**: Adam
- **Learning Rate**: 0.001
- **Batch size**: 8
- **Loss**: Cross Entropy

Due to time constraints, I had to stop the training of the model after the **8th epoch**.

The mean loss of the last epoch has been of 1.3579.

Darknet-53 - Testing

The model has been tested on 100,000 images, with 100 samples for each class.

The metric used for the evaluation is the Accuracy, more specifically the **TOP-1** and the **TOP-5** accuracy, with the following results:

- **TOP-1**: 0.10123
- **TOP-5**: 0.22513

PyTorch implementation of YOLOv3

YOLOv3 - NuImages Dataset

The scenario in which the YOLOv3 will be used is the **Object Detection for Driving Vehicles**.

The dataset used is the **NuImages** dataset, which contains:

- 93,000 annotated images
- 800,000 2-D bounding boxes for foreground objects

The most labeled class objects in the dataset are:

- Car (36.05%)
- Pedestrian (21.61%)

YOLOv3 - Model

```
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=255
activation=linear


[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=80
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
```

YOLOv3 - Model

```
yolo_architecture = [
    (32, 3, 1),
    (64, 3, 2),
    ["residual", 1],
    (128, 3, 2),
    ["residual", 2],
    (256, 3, 2),
    ["residualYolo", 8],
    # first yolo route
    (512, 3, 2),
    ["residualYolo", 8],
    # second yolo route
    (1024, 3, 2),
    ["residual", 4],
    # third yolo route
    ["yolo", 1024],
    ["yolo", 512],
    ["yolo", 256],
]
```

YOLOv3 - Training

Before training YOLO, some “hyper parameters” need to be defined:

- Grid sizes
- Anchor boxes width and height values

The model trained for almost **4 days** with the following parameters:

- **Optimizer**: Adam
- **Learning Rate**: 0.001
- **Batch size**: 8
- **Loss**: YOLO loss

YOLOv3 - YOLO loss

YOLO loss is the sum of the following individual loss values:

- **Anchor boxes where there is an object**:
  - **Objectness Loss**: Binary Cross Entropy of the objectness score.

  - **Coordinates Loss**: Mean Squared Error between the coordinate values tx, ty, tw, th.

  - **Class Loss**: Cross Entropy on the class prediction.

- **Anchor boxes where there is no object**:

  - **No Object Loss**: Binary Cross Entropy on the objectness score.

YOLOv3 - Training

Due to time constraints the YOLO model has been stopped at **epoch 28**, with the mean of the loss in the last epoch of 3.3306.

The model clearly has not finished to be trained, which has been resulted in poor performances.

Nevertheless, some detections are still valid, more precisely, the class in which the model performs better is Car.

This is showed in a small **web application** that I've created to test the model on input videos, with a particular attention on the on the number of **FPS** generated by YOLOv3.

Thanks for your attention!