Introduction

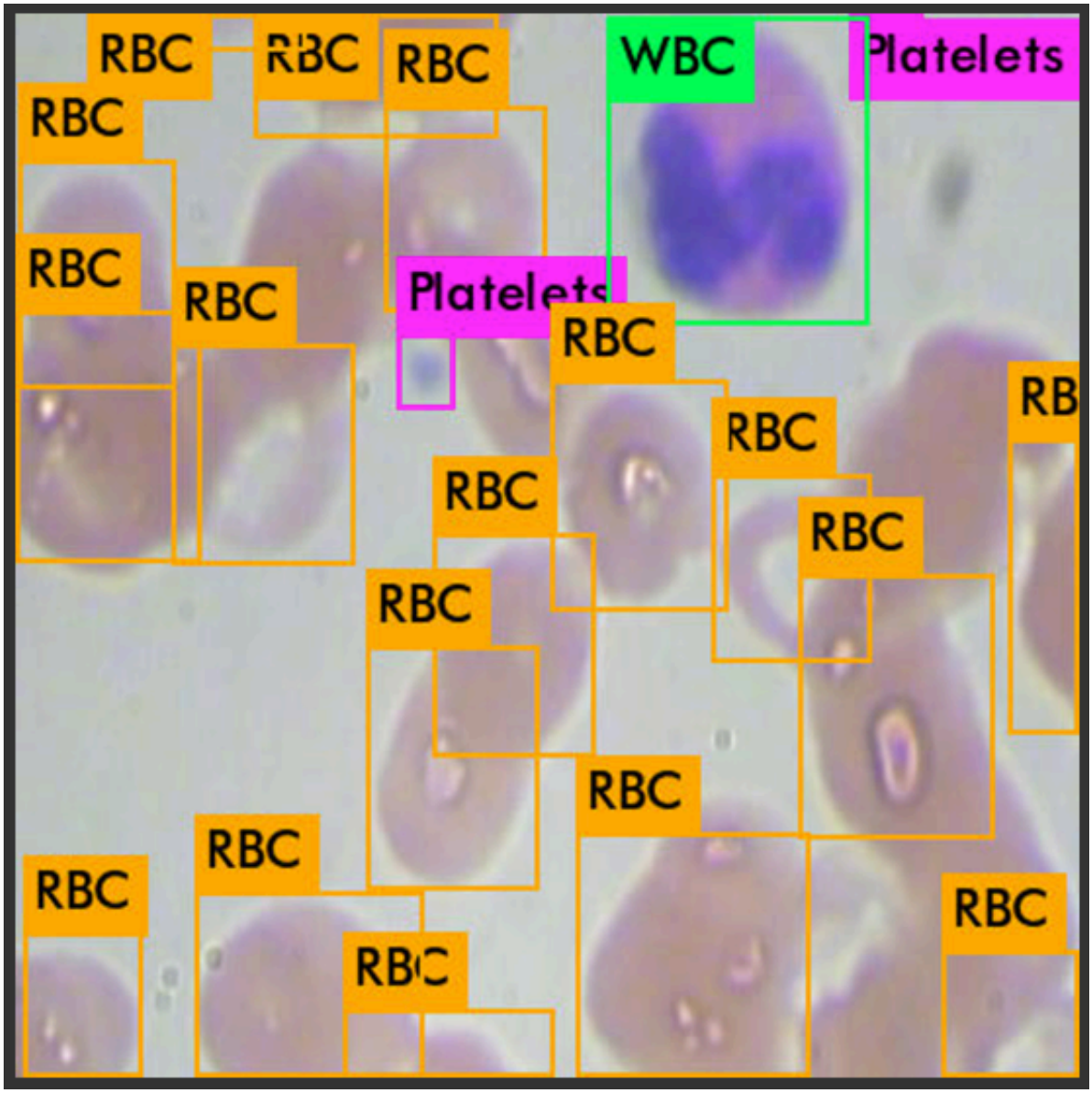
In this notebook, we implement the tiny version of YOLOv4 for training on your own dataset, YOLOv4 tiny.

We also recommend reading our blog post on Training YOLOv4 on custom data side by side.

We will take the following steps to implement YOLOv4 on our custom data:

- Configure our GPU environment on Google Colab
- Install the Darknet YOLOv4 training environment
- Download our custom dataset for YOLOv4 and set up directories
- Configure a custom YOLOv4 training config file for Darknet
- Train our custom YOLOv4 object detector
- Reload YOLOv4 trained weights and make inference on test images

When you are done you will have a custom detector that you can use. It will make inference like this:



Configuring cuDNN on Colab for YOLOv4

```
# CUDA: Let's check that Nvidia CUDA drivers are already pre-installed and which
!/usr/local/cuda/bin/nvcc --version
# We need to install the correct cuDNN according to this output
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2024 NVIDIA Corporation
Built on Thu_Jun__6_02:18:23_PDT_2024
Cuda compilation tools, release 12.5, V12.5.82
Build cuda_12.5.r12.5/compiler.34385749_0
```

```
#take a look at the kind of GPU we have
!nvidia-smi
```

```
Sat Mar  8 01:20:17 2025
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Versio |
|-----------------------------------------+------------------------+----------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util |
|                                         |                        |          |
|=========================================+========================+==========|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |          |
| N/A   56C    P8             12W /   70W |       0MiB /  15360MiB |      0%  |
|                                         |                        |          |
+-----------------------------------------+------------------------+----------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                             |
|        ID   ID                                                              |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
# This cell ensures you have the correct architecture for your respective GPU
# If you command is not found, look through these GPUs, find the respective
# GPU and add them to the archTypes dictionary

# Tesla V100
# ARCH= -gencode arch=compute_70,code=[sm_70,compute_70]

# Tesla K80
# ARCH= -gencode arch=compute_37,code=sm_37

# GeForce RTX 2080 Ti, RTX 2080, RTX 2070, Quadro RTX 8000, Quadro RTX 6000, Quad
# ARCH= -gencode arch=compute_75,code=[sm_75,compute_75]

# Jetson XAVIER
# ARCH= -gencode arch=compute_72,code=[sm_72,compute_72]

# GTX 1080, GTX 1070, GTX 1060, GTX 1050, GTX 1030, Titan Xp, Tesla P40, Tesla P4
# ARCH= -gencode arch=compute_61,code=sm_61

# GP100/Tesla P100 - DGX-1
# ARCH= -gencode arch=compute_60,code=sm_60

# For Jetson TX1, Tegra X1, DRIVE CX, DRIVE PX - uncomment:
# ARCH= -gencode arch=compute_53,code=[sm_53,compute_53]
```

```
# For Jetson Tx2 or Drive-PX2 uncomment:
# ARCH= -gencode arch=compute_62,code=[sm_62,compute_62]
import os
os.environ['GPU_TYPE'] = str(os.popen('nvidia-smi --query-gpu=name --format=csv,n

def getGPUArch(argument):
  try:
    argument = argument.strip()
    # All Colab GPUs
    archTypes = {
        "Tesla V100-SXM2-16GB": "-gencode arch=compute_70,code=[sm_70,compute_70]
        "Tesla K80": "-gencode arch=compute_37,code=sm_37",
        "Tesla T4": "-gencode arch=compute_75,code=[sm_75,compute_75]",
        "Tesla P40": "-gencode arch=compute_61,code=sm_61",
        "Tesla P4": "-gencode arch=compute_61,code=sm_61",
        "Tesla P100-PCIE-16GB": "-gencode arch=compute_60,code=sm_60"

      }
    return archTypes[argument]
  except KeyError:
    return "GPU must be added to GPU Commands"
os.environ['ARCH_VALUE'] = getGPUArch(os.environ['GPU_TYPE'])

print("GPU Type: " + os.environ['GPU_TYPE'])
print("ARCH Value: " + os.environ['ARCH_VALUE'])
```

```
GPU Type: Tesla T4

ARCH Value: -gencode arch=compute_75,code=[sm_75,compute_75]
```

Installing Darknet for YOLOv4 on Colab

```
%cd /content/
%rm -rf darknet
```

```
/content
```

```
#we clone the fork of darknet maintained by roboflow
#small changes have been made to configure darknet for training
!git clone https://github.com/roboflow-ai/darknet.git
```

```
Cloning into 'darknet'...
remote: Enumerating objects: 13289, done.
remote: Total 13289 (delta 0), reused 0 (delta 0), pack-reused 13289 (from 1)
Receiving objects: 100% (13289/13289), 12.17 MiB | 4.40 MiB/s, done.
^C
```

```
#install environment from the Makefile
%cd /content/darknet/
# compute_37, sm_37 for Tesla K80
# compute_75, sm_75 for Tesla T4
# !sed -i 's/ARCH= -gencode arch=compute_60,code=sm_60/ARCH= -gencode arch=comput
```

```
#install environment from the Makefile
#note if you are on Colab Pro this works on a P100 GPU
#if you are on Colab free, you may need to change the Makefile for the K80 GPU
#this goes for any GPU, you need to change the Makefile to inform darknet which G
!sed -i 's/OPENCV=0/OPENCV=1/g' Makefile
!sed -i 's/GPU=0/GPU=1/g' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/g' Makefile
!sed -i "s/ARCH= -gencode arch=compute_60,code=sm_60/ARCH= ${ARCH_VALUE}/g" Makef
!make
```

```
[Errno 2] No such file or directory: '/content/darknet/'
/content
sed: can't read Makefile: No such file or directory
sed: can't read Makefile: No such file or directory
sed: can't read Makefile: No such file or directory
sed: can't read Makefile: No such file or directory
make: *** No targets specified and no makefile found.  Stop.
```

```
#download the newly released yolov4-tiny weights
%cd /content/darknet
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/y
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/y
```

```
[Errno 2] No such file or directory: '/content/darknet'
/content
--2025-03-08 01:20:23--  https://github.com/AlexeyAB/darknet/releases/downloa
Resolving github.com (github.com)... 140.82.113.3
Connecting to github.com (github.com)|140.82.113.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asse
--2025-03-08 01:20:23--  https://objects.githubusercontent.com/github-product
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|18
HTTP request sent, awaiting response... 200 OK
Length: 24251276 (23M) [application/octet-stream]
Saving to: ‘yolov4-tiny.weights’

yolov4-tiny.weights 100%[===================>]  23.13M  64.8MB/s    in 0.4s

2025-03-08 01:20:24 (64.8 MB/s) - ‘yolov4-tiny.weights’ saved [24251276/24251

--2025-03-08 01:20:24--  https://github.com/AlexeyAB/darknet/releases/downloa
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asse
--2025-03-08 01:20:24--  https://objects.githubusercontent.com/github-product
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|18
HTTP request sent, awaiting response... 200 OK
Length: 19789716 (19M) [application/octet-stream]
Saving to: ‘yolov4-tiny.conv.29’

yolov4-tiny.conv.29 100%[===================>]  18.87M  80.5MB/s    in 0.2s

2025-03-08 01:20:25 (80.5 MB/s) - ‘yolov4-tiny.conv.29’ saved [19789716/19789
```

Set up Custom Dataset for YOLOv4

We'll use Roboflow to convert our dataset from any format to the YOLO Darknet format.

1. To do so, create a free Roboflow account.
2. Upload your images and their annotations (in any format: VOC XML, COCO JSON, TensorFlow CSV, etc).
3. Apply preprocessing and augmentation steps you may like. We recommend at least `auto-orient` and a `resize` to 416x416. Generate your dataset.
4. Export your dataset in the **YOLO Darknet format**.
5. Copy your download link, and paste it below.

See our blog post for greater detail.

In this example, I used the open source BCCD Dataset. (You can `fork` it to your Roboflow account to follow along.)

```
#follow the link below to get your download code from from Roboflow
!pip install -q roboflow
from roboflow import Roboflow
rf = Roboflow(model_format="darknet", notebook="roboflow-yolov4-tiny")
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 83.1/83.1 kB 2.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 66.8/66.8 kB 6.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.9/49.9 MB 19.6 MB/s eta 0:00:0(
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-8-05fac0eb4601> in <cell line: 0>()
      2 get_ipython().system('pip install -q roboflow')
      3 from roboflow import Roboflow
----> 4 rf = Roboflow(model_format="darknet", notebook="roboflow-yolov4-
tiny")

                                 2 frames
/usr/local/lib/python3.11/dist-packages/roboflow/__init__.py in
check_key(api_key, model, notebook, num_retries)
     21 def check_key(api_key, model, notebook, num_retries=0):
     22     if not isinstance(api_key, str):
---> 23         raise RuntimeError(
     24             "API Key is of Incorrect Type \n Expected Type: " +
str(str) + "\n Input Type: " + str(type(api_key))
     25         )

RuntimeError: API Key is of Incorrect Type
 Expected Type: <class 'str'>
 Input Type: <class 'NoneType'>
```

Next steps: Explain error

```
# from roboflow import Roboflow
# rf = Roboflow(api_key="YOUR_API_KEY")
# project = rf.workspace().project("YOUR_PROJECT")
# dataset = project.version("YOUR_VERSION").download("darknet")
```

```
#Set up training file directories for custom dataset
%cd /content/darknet/
%cp {dataset.location}/train/_darknet.labels data/obj.names
%mkdir data/obj
#copy image and labels
%cp {dataset.location}/train/*.jpg data/obj/
%cp {dataset.location}/valid/*.jpg data/obj/

%cp {dataset.location}/train/*.txt data/obj/
%cp {dataset.location}/valid/*.txt data/obj/

with open('data/obj.data', 'w') as out:
  out.write('classes = 3\n')
  out.write('train = data/train.txt\n')
  out.write('valid = data/valid.txt\n')
  out.write('names = data/obj.names\n')
  out.write('backup = backup/')

#write train file (just the image list)
import os

with open('data/train.txt', 'w') as out:
  for img in [f for f in os.listdir(dataset.location + '/train') if f.endswith('j
    out.write('data/obj/' + img + '\n')

#write the valid file (just the image list)
import os

with open('data/valid.txt', 'w') as out:
  for img in [f for f in os.listdir(dataset.location + '/valid') if f.endswith('j
    out.write('data/obj/' + img + '\n')
```

Write Custom Training Config for YOLOv4

```
#we build config dynamically based on number of classes
#we build iteratively from base config files. This is the same file shape as cfg/
def file_len(fname):
  with open(fname) as f:
    for i, l in enumerate(f):
      pass
  return i + 1

num_classes = file_len(dataset.location + '/train/_darknet.labels')
max_batches = num_classes*2000
steps1 = .8 * max_batches
steps2 = .9 * max_batches
steps_str = str(steps1)+','+str(steps2)
num_filters = (num_classes + 5) * 3


print("writing config for a custom YOLOv4 detector detecting number of classes: "

#Instructions from the darknet repo
#change line max_batches to (classes*2000 but not less than number of training im
#change line steps to 80% and 90% of max_batches, f.e. steps=4800,5400
if os.path.exists('./cfg/custom-yolov4-tiny-detector.cfg'): os.remove('./cfg/cust


#customize iPython writefile so we can write variables
from IPython.core.magic import register_line_cell_magic

@register_line_cell_magic
def writetemplate(line, cell):
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))


%%writetemplate ./cfg/custom-yolov4-tiny-detector.cfg
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=24
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
```

```
exposure = 1.5
hue=.1

learning_rate=0.00261
burn_in=1000
max_batches = {max_batches}
policy=steps
steps={steps_str}
scales=.1,.1

[convolutional]
batch_normalize=1
filters=32
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[route]
layers=-1
groups=2
group_id=1

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

[convolutional]
```

```
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

[route]
layers = -1,-2

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[route]
layers = -6,-1

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[route]
layers=-1
groups=2
group_id=1

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
```

```
filters=64
size=3
stride=1
pad=1
activation=leaky

[route]
layers = -1,-2

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

[route]
layers = -6,-1

[maxpool]
size=2
stride=2

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

[route]
layers=-1
groups=2
group_id=1

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
```

```
stride=1
pad=1
activation=leaky

[route]
layers = -1,-2

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

[route]
layers = -6,-1

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

##################################

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

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky
```

```
[convolutional]
size=1
stride=1
pad=1
filters={num_filters}
activation=linear



[yolo]
mask = 3,4,5
anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
classes={num_classes}
num=6
jitter=.3
scale_x_y = 1.05
cls_normalizer=1.0
iou_normalizer=0.07
iou_loss=ciou
ignore_thresh = .7
truth_thresh = 1
random=0
nms_kind=greedynms
beta_nms=0.6

[route]
layers = -4

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

[upsample]
stride=2

[route]
layers = -1, 23

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

```
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters={num_filters}
activation=linear

[yolo]
mask = 1,2,3
anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
classes={num_classes}
num=6
jitter=.3
scale_x_y = 1.05
cls_normalizer=1.0
iou_normalizer=0.07
iou_loss=ciou
ignore_thresh = .7
truth_thresh = 1
random=0
nms_kind=greedynms
beta_nms=0.6


#here is the file that was just written.
#you may consider adjusting certain things

#like the number of subdivisions 64 runs faster but Colab GPU may not be big enou
#if Colab GPU memory is too small, you will need to adjust subdivisions to 16
%cat cfg/custom-yolov4-tiny-detector.cfg
```

Train Custom YOLOv4 Detector

```
!./darknet detector train data/obj.data cfg/custom-yolov4-tiny-detector.cfg yolov
#If you get CUDA out of memory adjust subdivisions above!
#adjust max batches down for shorter training above
```

Infer Custom Objects with Saved YOLOv4 Weights

```
#define utility function
def imShow(path):
  import cv2
  import matplotlib.pyplot as plt
  %matplotlib inline

  image = cv2.imread(path)
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER

  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  #plt.rcParams['figure.figsize'] = [10, 5]
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()


#check if weigths have saved yet
#backup houses the last weights for our detector
#(file yolo-obj_last.weights will be saved to the build\darknet\x64\backup\ for e
#(file yolo-obj_xxxx.weights will be saved to the build\darknet\x64\backup\ for e
#After training is complete - get result yolo-obj_final.weights from path build\d
!ls backup
#if it is empty you haven't trained for long enough yet, you need to train for at


#coco.names is hardcoded somewhere in the detector
%cp data/obj.names data/coco.names


#/test has images that we can test our detector on
test_images = [f for f in os.listdir('test') if f.endswith('.jpg')]
import random
img_path = "test/" + random.choice(test_images);

#test out our detector!
!./darknet detect cfg/custom-yolov4-tiny-detector.cfg backup/custom-yolov4-tiny-d
imShow('/content/darknet/predictions.jpg')
```