دانشگاه زنجان

گزارش پروژه پایانی درس هوش مصنوعی
تشخیص علائم راهنمایی و رانندگی
کیان صحافی

در مرحله اول تمامی عکس های مد نظر با ابزار LabelImg لیبل زده شدند که این پروسه زمانبر ترین قسمت پروژه بود. که متشکل از ۹۷۰ عکس لیبل زده شده است که کل این دیتاست را به دو قسمت train و validation تقسیم کردیم که در قسمت train کل داده ها (لیبل ها) را به شبکه عصبی بدهیم تا wها آپدیت شوند و در نهایت با داده ها (لیبل ها) validation بسنجیم که چقدر دقت شبکه عصبی ما بالا رفته است.
کل دیتا ست ما آپلود شده در سایت kaggle، به ادرس زیر است:
https://www.kaggle.com/datasets/kiansahafi/traffic-sign-recognition

و کد کامل در سایت kaggle، به ادرس زیر است که در ادامه با هم بررسی خواهیم کرد:
https://www.kaggle.com/kiansahafi/project-ai-object-detection/edit

---

در قسمت تمامی قسمت های نوت بوک این پروژه را مورد بررسی قرار خواهیم داد

```
pip install --upgrade pip
```

```
# !git clone https://github.com/ultralytics/yolov5
!pip install -U -r yolov5/requirements.txt  # install dependencies
```

در قسمت های بالا فقط dependency های مورد نیاز را نصب کرده ایم و همانطور که گفته شده از ابزار YOLO در این پروژه استفاده خواهیم کرد که یکی از بهترین الگوریتم های ابجکت دیتکشن(object detection) رو داره.

```
!pip install torch torchvision -f https://download.pytorch.org/whl/torch_stable.html
```

```
%cd /kaggle/working/yolov5
!ls
```

```
/kaggle/working/yolov5
CITATION.cff     benchmarks.py  exp3             setup.cfg       yolov5
CONTRIBUTING.md  classify       export.py        state.db        yolov5s.pt
LICENSE          data           hubconf.py       train.py
README.md        detect.py      models           tutorial.ipynb
README.zh-CN.md  exp            requirements.txt  utils
__pycache__      exp2           segment          val.py
```

در قسمت بالا ابزار های torch و torchvision رو نصب میکنیم، و در قسمت پایین از وجود کلاس های YOLO اطمینان کسب میکنیم.

در قسمت بالا از ورژن torch اطمینان حاصل میکنیم و اطلاعات GPU را پرینت میکنیم.

دانشگاه زنجان

```
import torch
from IPython.display import Image  # for displaying images

print('Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
```

```
Using torch 2.0.1+cu117 _CudaDeviceProperties(name='Tesla P100-PCIE-16GB', major=6, minor=0, total_memory=16280MB, multi_processor_count=56)
```

در قسمت بالا ما باید فایلی داشته باشیم که تعداد علامت هایی که قرار است آموزش دیده شود و مشخصات آنها و مسیر

```
%cd /kaggle/input/datayaml-dataset
%cat data.yaml
```

```
/kaggle/input/datayaml-dataset

train: /kaggle/input/traffic-sign-recognition/training/training/images

val: /kaggle/input/traffic-sign-recognition/valid/valid/images


nc: 12

names: ['Dont Enter','Keep Right','Max Speed 40','Max Speed 60', 'No Horn', 'No Left Turn', 'Keep Straight', 'No Right Turn', 'No U Turn', 'Parking', 'Bicycle', 'Cro
ss Walk']
```

validaiton و train در آن مشخص شده باشد که ما در اینجا فایل data.yaml را داریم که اطلاعات داخل ان بشکل بالا است.
در قسمت بالا ما دیتای داخل data.yaml را میخونیم و پارامتر مربوط به nc را هم میخونیم که نشون دهنده کلاس ها

```
import yaml
with open("data.yaml", 'r') as stream:
    num_classes = str(yaml.safe_load(stream)['nc'])
```

هستش.

```
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()))
```

در قسمت بالا یک سری دستورات خاصی میسازد برای استفاده در IPython که اجازه میده محتوای یک سلول رو داخل یک فایل ذخیره کنیم.

دانشگاه زنجان

```
with open(r'data.yaml') as file:
    # The FullLoader parameter handles the conversion from YAML
    # scalar values to Python the dictionary format
    labels_list = yaml.load(file, Loader=yaml.FullLoader)

    label_names = labels_list['names']
```

+ Code + Markdown

```
print("Number of Classes are {}, whose labels are {} for this Object Detection project".format(num_classes,label_names))
```

```
Number of Classes are 12, whose labels are ['Dont Enter', 'Keep Right', 'Max Speed 40', 'Max Speed 60', 'No Horn', 'No Left Turn', 'Keep Straight', 'No Right Turn',
'No U Turn', 'Parking', 'Bicycle', 'Cross Walk'] for this Object Detection project
```

در قسمت بالا فایل data.yaml خونده میشه و تمام اسم لیبل ها و یا همون کلاس ها داخل وریبلی بنام label_names ریخته میشه. و پرینت.

```
%cat /kaggle/working/yolov5/models/yolov5s.yaml
```

```
# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

# Parameters
nc: 12  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]

# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

در کد روبرو ما ساختار دیفالت yolo5 رو میبینیم، که برای تمرین بیشتر ما میخواهیم خودمون معماری شبکه عصبی رو طراحی کنیم!

دانشگاه زنجان

```yaml
# Parameters
nc: {num_classes}  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]

# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

کد روبرو تنظیمات ما بر روی معماری yolo خواهد بود که در این کد همانطور که مشخص است عمق و عرض شبکه عصبی مشخص شده است، و در مرحله بعد anchor ها هستند که نشان دهنده ی جعبه های لنگر!(anchor boxes) هستند که هر خط متشکل از چندین جعبه است که عدد اول و دوم به ترتیب عرض و ارتفاع این جعبه ها را در اسکیل اول (first scale) نشان میدهد و عدد سوم و چهارم به ترتیب عرض و ارتفاع این جعبه ها را در اسکیل دوم و ... .

در قسمت بعدی یعنی Backbone ستون معماری yolo را تعریف میکند که متشکل از یک سری لایه است که با یک لیستی از اعداد تعریف شده اند. که هر لایه میتواند از این سه حالت باشد: Conv, C3, SPPF

در قسمت بعدی یعنی قسمت Head مثل قسمت قبلی است یعنی این قسمت هم متشکل از یک سری لایه است ولی این قسمت وظیفه پروسس کردن فیچرهایی که در قسمت قبلی بدست آمده است را به عهده دارد. به علاوه پیشبینی کلاس ها و object bounding box ها و دادن نمره اعتماد بنفس (confidence score) ها است.

```
%cd /kaggle/working/yolov5
!python train.py --img 360 --batch 50 --epochs 100 --data '/kaggle/input/datayaml-dataset/data.yaml' --cfg /kaggle/working/yolov5/models/custom_yolov5s.yaml --cache --project ''
```

در نهایت مرحله آموزش دیتا رو شروع میکنیم، با کیفیت عکس ۳۶۰ ،با بچ های ۵۰ تایی و با ۱۰۰ بار همین داده هارو به شبکه عصبی میدهیم تا w ها آپدیت شوند.

دانشگاه زنجان

```
      0/99      4.66G      0.118    0.01011    0.06854         41        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58   0.000597      0.144    0.00064    0.00016

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      1/99       4.66G    0.06971    0.01058    0.06042         20        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58    0.00362      0.788     0.0201    0.00767

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      2/99       4.66G    0.05552   0.009083    0.05687         39        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.924      0.081      0.118     0.0413
```

همانطور که پیداست در چندین epoch اول دقت ما خیلی پایین است و این دقت در epoch های بیشتر بالاتر میرود.

```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      96/99      4.66G     0.0195   0.002667   0.005139         28        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.885      0.789      0.795      0.654

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      97/99      4.66G    0.01852   0.002504   0.005179         48        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.889      0.803      0.797      0.638

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      98/99      4.66G    0.01889   0.002603   0.004893         58        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.862      0.805      0.797      0.636

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      99/99      4.66G    0.01893   0.002637   0.004898         30        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58       0.89      0.788      0.801      0.652
```

در چهار epoch اخر تغییر بسیار قابل توجهی وجود ندارد شاید بعلت overfitting است. و دقت هم خیلی بالا نیست شاید بعلت کم بودن دیتا ست میباشد.

```
                 all        101         58       0.85      0.703      0.863      0.674
           Dont Enter        101          5          1      0.844      0.995      0.653
           Keep Right        101          1      0.462          1      0.995      0.796
        Max Speed 40         101         17          1       0.66      0.978       0.76
        Max Speed 60         101          3       0.71          1      0.995      0.852
             No Horn         101          6      0.991      0.833      0.955      0.752
        No Left Turn         101          1      0.912          1      0.995      0.895
       No Right Turn         101          4          1      0.306      0.995      0.714
          No U Turn          101          2      0.426        0.5      0.745      0.571
             Bicycle         101          6          1          0          0          0
          Cross Walk         101         13          1       0.89      0.979      0.743
```

آمار کلی کلاس ها هم بصورت بالا است.

```
lov5
 360 --batch 50 --epochs 100 --data '/kaggle/input/datayaml-dataset/data.yaml' --cfg /kaggle/working/yolov5/models/custom_yolov5s.yaml --weights '/kaggle/working/yolov5/exp3/weig
```

یکبار دیگر با وزن های شبکه عصبی قبلی که ذخیره کرده بودیم دوباره عملیات train رو انجام دادیم که نتایج به صورت زیر است.

دانشگاه زنجان

```
      0/99      4.66G    0.02954   0.002797   0.007932         41        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.809      0.817      0.839      0.612

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      1/99      4.66G    0.03024   0.002748   0.008043         20        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.802      0.781      0.787      0.598

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      2/99      4.66G    0.03081   0.003091   0.009908         39        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.816      0.663      0.802      0.603

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      3/99      4.66G    0.03219   0.003373     0.0117         30        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.672      0.686      0.708       0.53
```

همینطور که مشخص است در epoch های اول دقت ما بالاست!

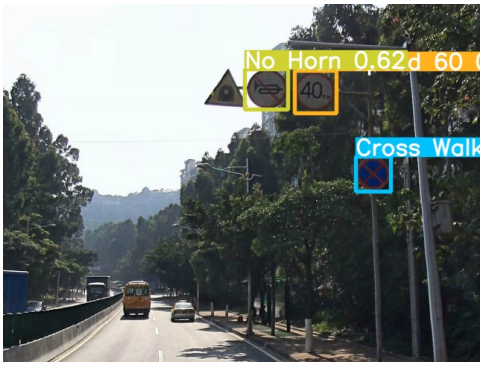
```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      96/99     4.66G    0.01767    0.00234   0.003017         28        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.861      0.837      0.798      0.646

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      97/99     4.66G     0.0172   0.002233   0.002824         48        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.873      0.822        0.8      0.651

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      98/99     4.66G    0.01761   0.002305    0.00283         58        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.876       0.82        0.8      0.627

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      99/99     4.66G    0.01747   0.002363   0.002966         30        384: 1
                 Class     Images  Instances          P          R      mAP50
                   all        101         58      0.874      0.821      0.798       0.64
```

ولی همینطور که مشخص است بعد از دادن وزن های شبکه عصبی قبلی تغییر زیادی بوجود نیامده است!

حالا بریم سراغ دیتا های تست:

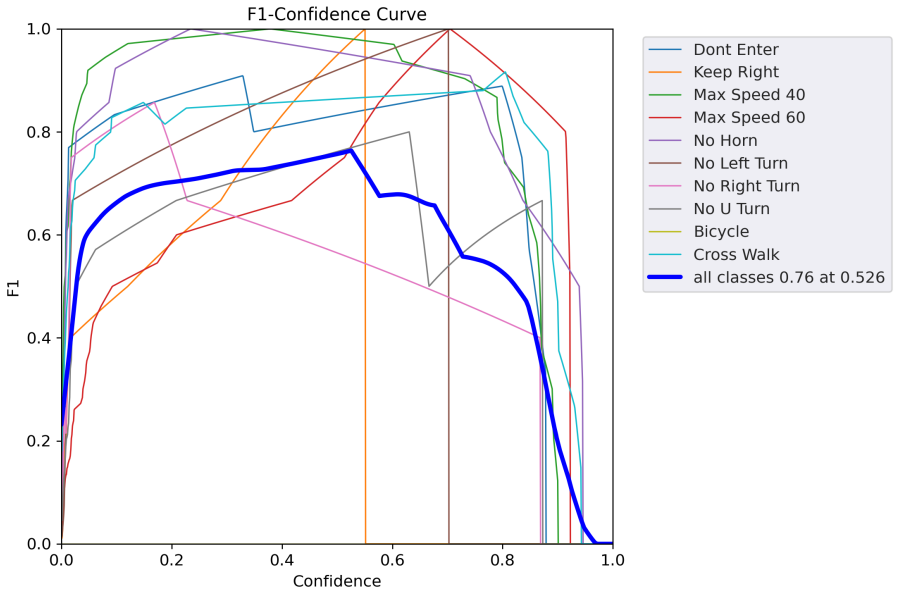
```
%cd /kaggle/working/yolov5
!python detect.py --weights '/kaggle/working/yolov5/exp3/weights/best.pt' --img 360 --conf 0.4 --source '/kaggle/in
```

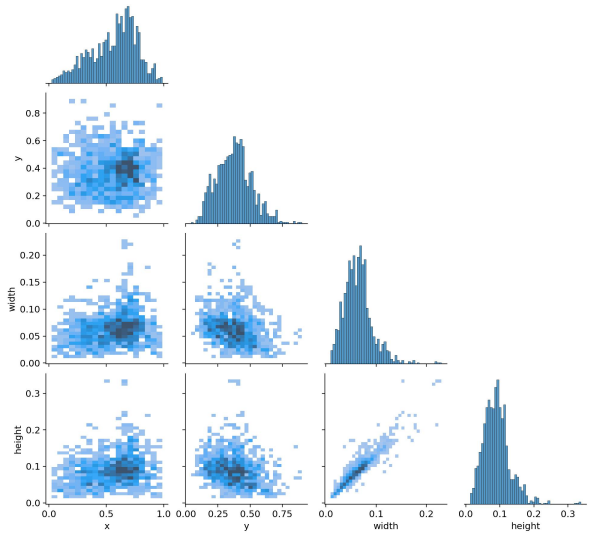
که در این تکه کد ما شبکه عصبی اموزش دیده رو مورد بررسی قرار میدیم.
بعضی نتایج:

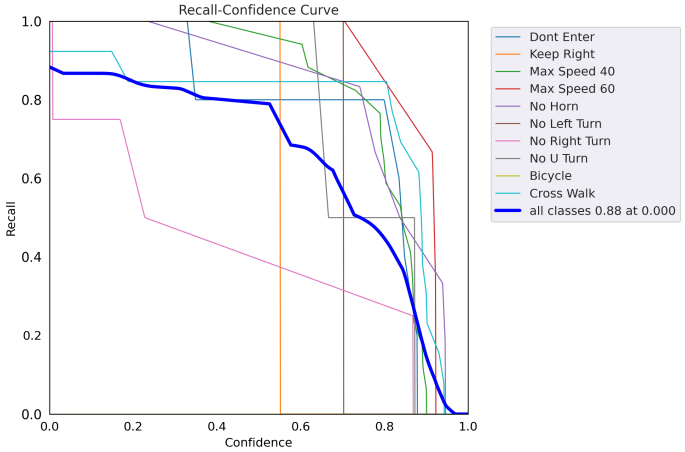


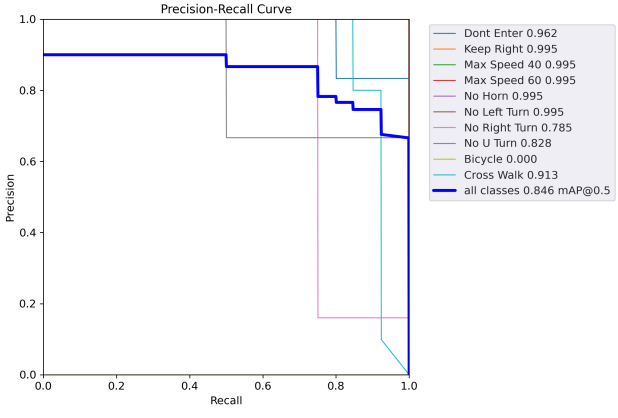
دانشگاه زنجان

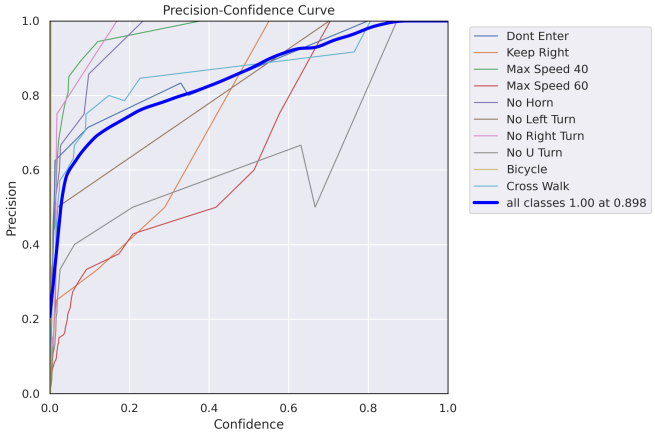
که در ادامه اماری که توسط yolo جنریت شده است رو قرار میدم:



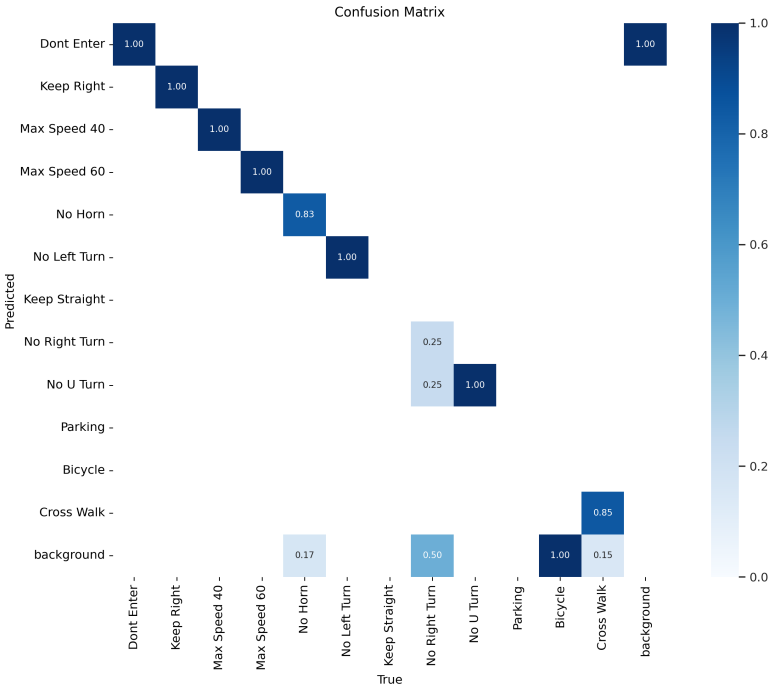


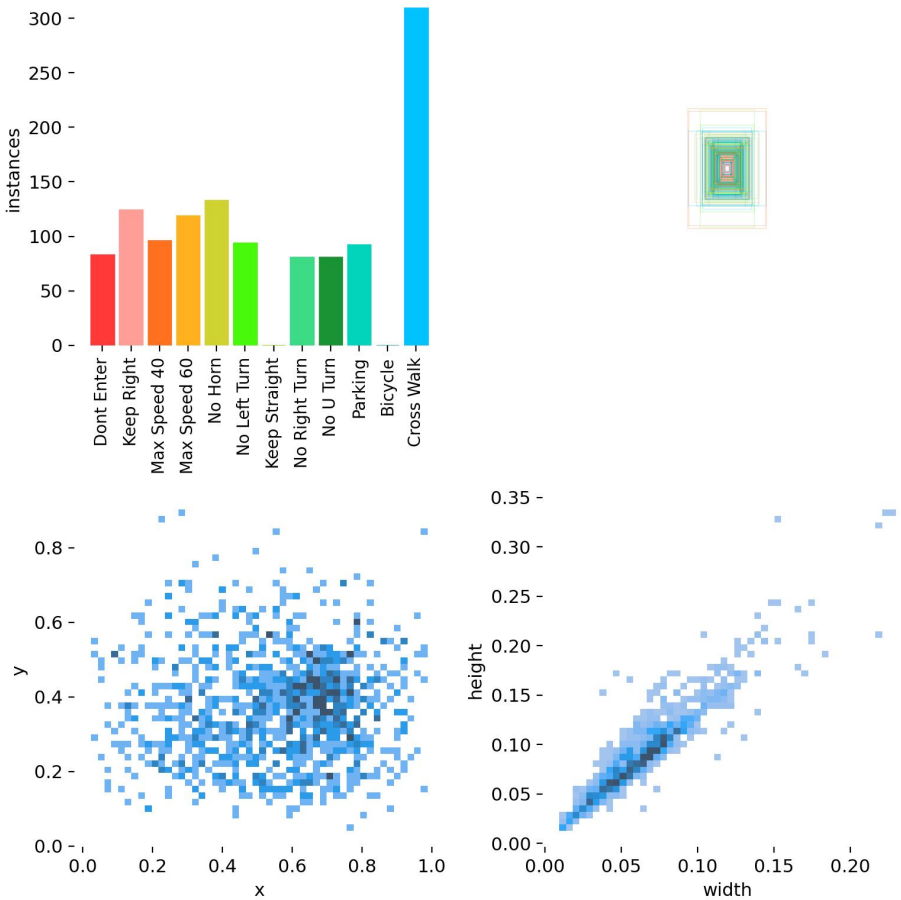






دانشگاه زنجان





دانشگاه زنجان