

RNIC 建模指南

文档版本：1.0

文档作者：qq600585

目录

什么是 RNIC?	2
RNIC 和 MMOD(MaxMagrginObjectDetector)	3
RNIC 和 SS(semantic segmentation)	3
RNIC 和 FACE-MDNN	3
单独对待 RNIC 训练	4
确认 ZAI 的 Key 是否可以正常使用	4
Cuda 准备工作	5
准备图片数据集	6
训练 RNIC	6
二级分类	7
5 分钟后，2 级分的汽车轮毂被训练完成	8
运行 Demo 测试模型	8

什么是 RNIC?

RNIC, ResnetImageClassifier, 中文翻译: 基于残差网络的图片分类器, *模式识别*。

得益于 GPU 的大规模处理能力, RNIC 在训练时可以逐渐学习图像的内容, RNIC 在工作时会将图像随机剪裁, 然后做拟合, 当达到拟合条件, 就会得到图像的内容的相似性, 通过大规模操作, RNIC 便具备了识别图像内容的能力: *模式识别*。

模式识别是专业化应用机器学习的前置工作, 诸如人脸识别, 我们需要场景中有人, 模式识别可以提供场景中有人模式返回。假如我们要做车辆识别, 模式识别也会是前置工作, 第一步是确认场景有车, 第二步就是确认汽车的型号, 甚至主驾副驾的性别。

RNIC 和 MMOD(MaxMagrinObjectDetector)

- 运行机制: RNIC 是提取图像内容的学习, MMOD 是反复来模拟图像尺度和光影的学习。
- 制作工艺: 两者无论在运作机制还是数据集的制作工艺都有所差异, 而差异恰恰能互补, 两者一旦结合起来进行商业化应用, 能满足 **万物识别** 需求: **模式识别>特征识别**。 **万物识别** 这一过程, 几乎可以覆盖 90% 社会各界对机器学习需求, MMOD 可以通过数据收敛, 得到检测对象标签, 并且可以明察秋毫的区分细节, 诸如 Iphone5 和 Iphone7, 细节在于边缘, 颜色, MMOD 能聪明的加以区分, 然后正确标注。另外, MMOD 只需要足够多的框体数据集, 打标签, 训练, 即可使用。RNIC 只需要足够描述实际场景的照片, 输入矩阵工具, 训练, 即可使用。两者在模型的制作工艺上都很简单, 效果美丽, 楚楚动人, 覆盖社会对机器学习需求面非常广阔。注意: 训练应用级的 **模式识别>特征识别** 对设备要求很 **苛刻**, 需要顶级配置设备才能投入实际训练, 这两者都需要大量图片做梯度收敛, 这会导致巨大的显存开销, 巨大的显存开销也意味着巨大投资。当模型被制作完成, **模式识别>特征识别** 运行载体则很平民化, 使用普通支持 CUDA 显卡即可运行。

RNIC 和 SS(semantic segmentation)

- 运行机制: SS 和 RNIC 在构建深度网络 Layer 上, 有一定的相似性, SS 多了图像的像素或则折线几何标注, RNIC 则只有 Label 标注, SS 的网络构建更加复杂。
- 制作工艺: SS 需要目标描绘标签+(目标折线几何构成 or 目标像素体构成), RNIC 只需要描绘标签。由于 SS 的制作工艺涉及到几何体+文本标注, SS 要复杂于 RNIC。因为 SS 需要折线几何系统构成, 数据集制作工艺尚未梳理完成。在近期 2019-3 月左右的版本会实现 SS 分割功能。我们试想一下, 从 RNIC 到 SS, 如果有好的想法, 请来信留言。

RNIC 和 FACE-MDNN

- 运行机制: 两者在 input net 前, RNIC 是做剪裁后 input, face-MDNN 是做贴片后 input。待完成后, 他们都会在相等分类下做拟合条件, 不符合就重新做 input, 这就是我们看到的失效步数。从网络 Layer 结构来说, 两者在做拟合层面有相似性, 在 resnet 预处理层面, 完全是两种机制。
- 制作工艺: 两者在实用时都会需要一个很大的数据集, RNIC 要用大数据集训练后才能做到识别各种图片的内容, FACE-MDNN 要学习成千上万的人脸才能准确认识目标。RNIC 的使用时独立场景, 多用于模式识别, FACE 的使用时连贯性场景, 多用于身份验证场景。我们试想一下, 从 RNIC 到人脸识别, 如果有好的想法, 请来信留言。

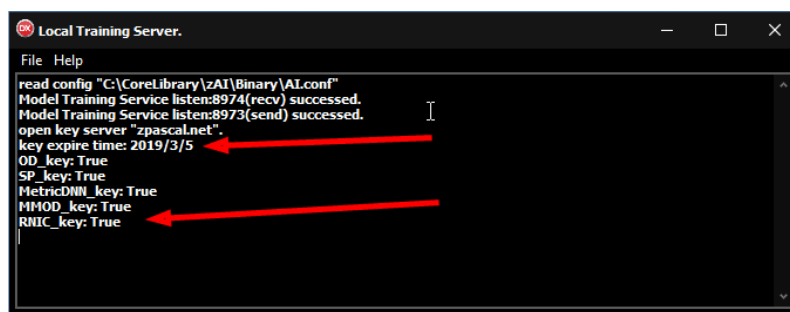
单独对待 RNIC 训练

由于 RNIC 是针对全幅图像内容的暴力训练，在内存，显存，GPU 等等开销上，属于小型超算级的运行需求，因此，zAI 对 RNIC 的训练定位是全手动方式，除了图片打包之外，没有自动化的工具，我们需要单独的制作工艺处理 RNIC 训练。

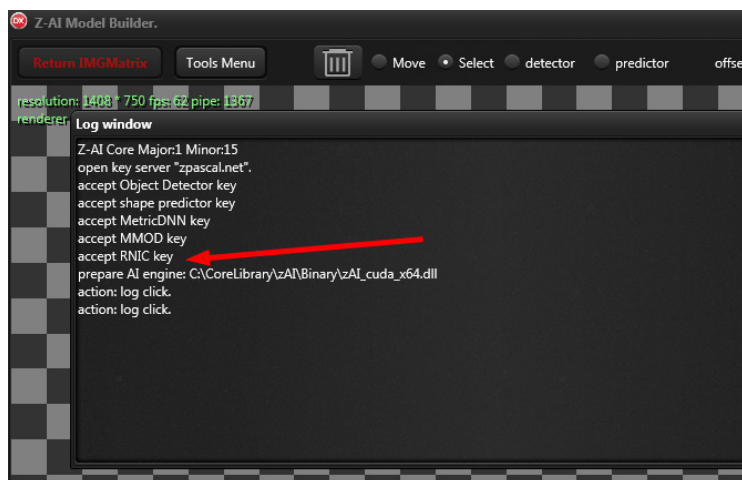
确认 ZAI 的 Key 是否可以正常使用

确保 RNIC 的 Key 可以正常使用

两个办法，在 LocalTrainingServer 确认 Key 状态



在 z_ai_model 确认 Key 状态



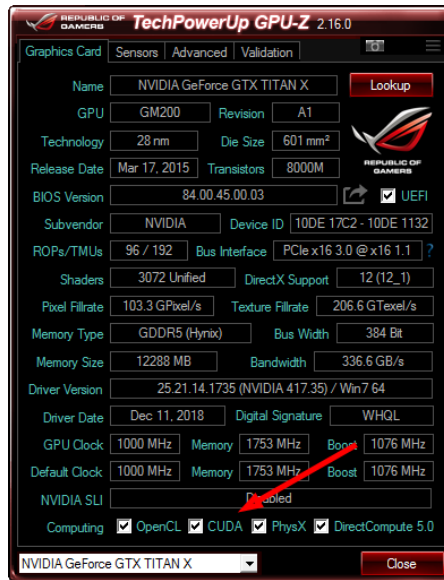
Cuda 准备工作

需要 cuda 硬件设备，参考文档，zAI 开发设备撰机指南.pdf

另一个是 gpu-z 检测器，它有两个版本，一个是华硕板皇的黑版，一个是普通版。主要差别是皮肤不一样。

<https://www.techpowerup.com/download/techpowerup-gpu-z/>

在 gpu-z 中确保显卡是支持 cuda 的

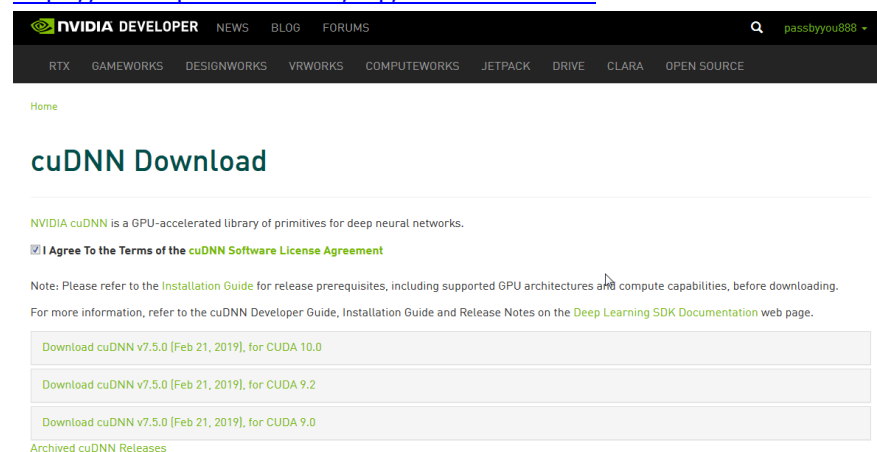


cuda 需要到 nvidia 的官方网站下载驱动，zAI 的 cuda 引擎使用 nvcc+cuda sdk10 构建，也支持 cuda9.2，如果是 cuda9.0 或更低的版本是不支持的。

<https://developer.nvidia.com/cuda-downloads>

另外我们还需要单独安装一个和 cuda 对应的 cuDNN 库，这个库比较大，且需要和 cuda 配套。提示：我们下载 cudnn 需要一个 nvidia 开发者账号，过程很简单，按提示操作即可。

<https://developer.nvidia.com/rdp/cudnn-download>



通过 zpascal.net 官网的 <https://zpascal.net> 也可以找到完整的 cuda sdk+cudnn 程序

准备图片数据集

首先，我们需要准备一批图片素材，然后以目录来分类。

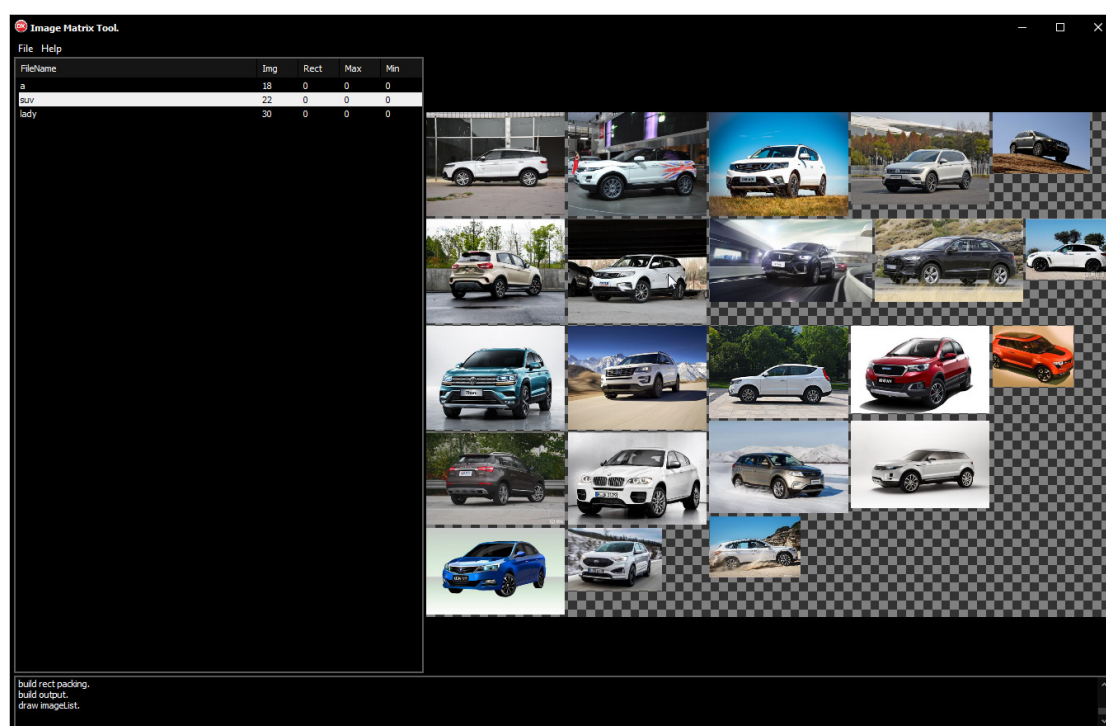
本文一共创建 3 个分类，

SUV，从百度采集的 SUV 车型特写，不区分正，侧，尾，顶

A，从百度采集的 A 级车型特写，不区分正，侧，尾，顶

LADY，从百度硬盘采集的一些女人特写，未露点

RNIC 单个分类不宜过大，也不宜过小，保持平均，实用建议单个分类照片在 100 张以上
如内存不够，可使用 script，用表达式给 scale(0.5) 进行减肥，不必每次都去动用 PS 做减肥。



训练 RNIC

处于测试的需要，本文使用 TrainingTool 训练，本文针对数据集构建了一个专属训练 Demo，位于目录 zAI\Demo\ResNetImgClassifier2，它能自动训练数据集文件 MiniImgClassifier.imgMat。并且提供测试功能。

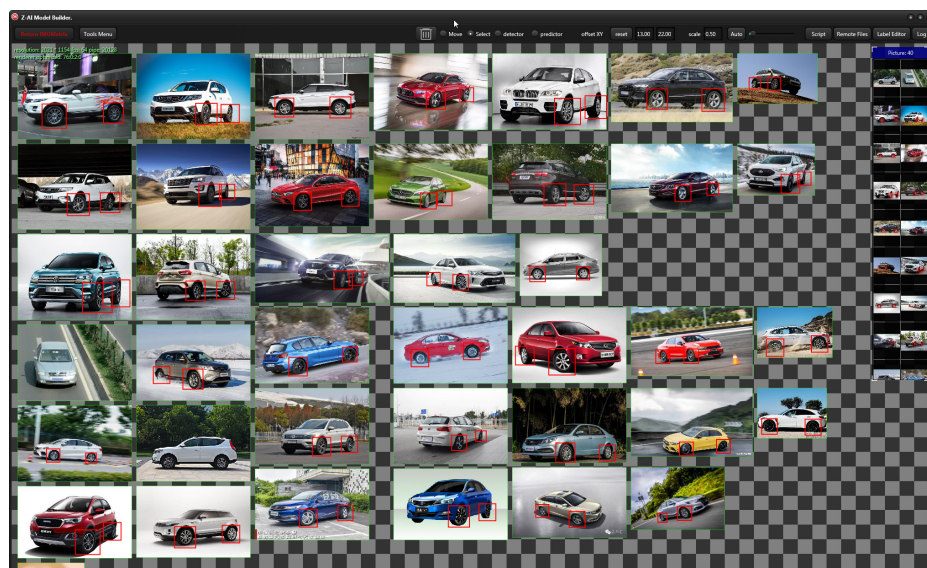
只需要将以上数据集保存成，MiniImgClassifier.imgMat，然后使用 ResNetImgClassifier2 来训练即可。旁边的按钮可用于测试分类结果。*数据集建大以后，亦可依照此方法进行训练+测试。*

二级分类

使用 z_ai_model 创建好汽车轮毂的框体标注数据集，保存，训练

本文以演示为主，数据集样本非常少，实际是做汽车轮毂，需要上千张真车数据集图片才能达到实用效果。请参考 MMOD 建模指南。

注意：在实际应用中，我们甚至可以通过轮毂形状，来建库，将它作为三级分类来处理，这样拍张照片，即可知道轮毂的制造厂商。



然后发送给服务器训练，因为数据集很小，2 分钟左右，loss 率已经到达 0.1，但是梯度还没开始收敛。我猜轮毂模型大概要训练 5 分钟吧。

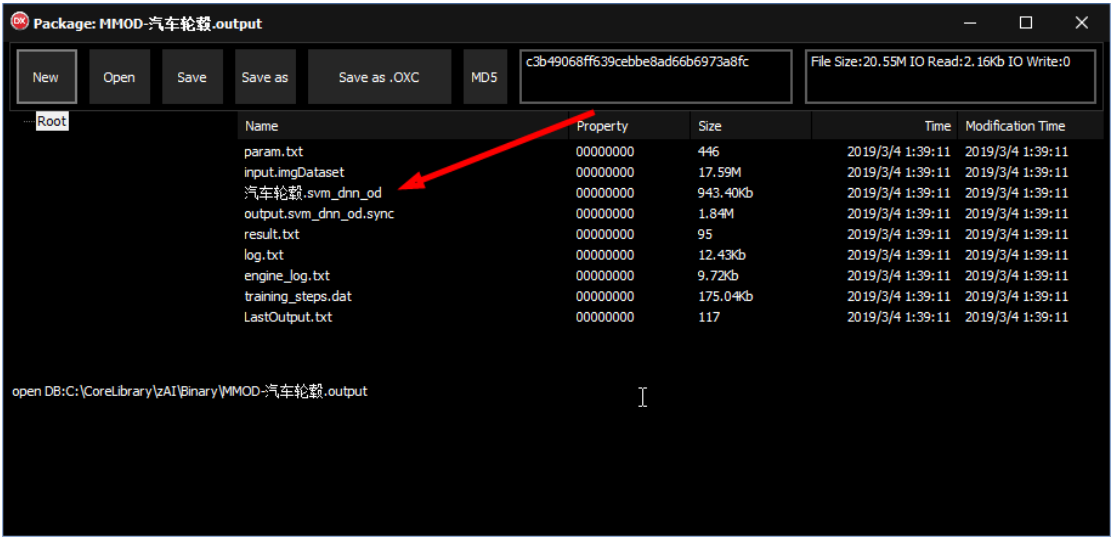
```
Local Training Server.
File Help
min_object_length_long_dim: 95
min_object_length_short_dim: 19
max_object_size: 0.7
background_crops_fraction: 0.5
translate_amount: 0.1

net structure:
layer<0>- loss_mmmod (detector: windows(100x100), loss per FA1, loss per miss1, truth match IOU thresh0.5, use_bounding_box_regressionfalse, overlaps_nms(0.227084,0.371474),
layer<1>- conv (num_filters=1, n=9, nc=9, stride_y=1, stride_x=1, padding_y=4, padding_x=4) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<2>- relu
layer<3>- bn_conv (num_filters=55, nr=5, nc=5, stride_y=1, stride_x=1, padding_y=2, padding_x=2) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<4>- conv (num_filters=55, nr=5, nc=5, stride_y=1, stride_x=1, padding_y=2, padding_x=2) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<5>- relu
layer<6>- bn_conv (num_filters=55, nr=5, nc=5, stride_y=1, stride_x=1, padding_y=2, padding_x=2) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<7>- conv (num_filters=55, nr=5, nc=5, stride_y=1, stride_x=1, padding_y=2, padding_x=2) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<8>- relu
layer<9>- bn_conv (num_filters=55, nr=5, nc=5, stride_y=1, stride_x=1, padding_y=2, padding_x=2) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<10>- conv (num_filters=55, nr=5, nc=5, stride_y=1, stride_x=1, padding_y=2, padding_x=2) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<11>- relu
layer<12>- bn_conv (num_filters=32, nr=5, nc=5, stride_y=2, stride_x=2, padding_y=0, padding_x=0) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<13>- conv (num_filters=32, nr=5, nc=5, stride_y=2, stride_x=2, padding_y=0, padding_x=0) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<14>- relu
layer<15>- bn_conv (num_filters=32, nr=5, nc=5, stride_y=2, stride_x=2, padding_y=0, padding_x=0) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<16>- conv (num_filters=32, nr=5, nc=5, stride_y=2, stride_x=2, padding_y=0, padding_x=0) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<17>- relu
layer<18>- bn_conv (num_filters=16, nr=5, nc=5, stride_y=2, stride_x=2, padding_y=0, padding_x=0) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<19>- conv (num_filters=16, nr=5, nc=5, stride_y=2, stride_x=2, padding_y=0, padding_x=0) learning_rate_mult=1 weight_decay_mult=1 bias_learning_rate_mult=1 bias_weight_decay_mult=1
layer<20>- input_rgb_image_pyramid(122.782,117.001,104.298) pyramid_padding=10 pyramid_outer_padding=11

step#: 0 learning rate: 0 average loss: 2.66317 steps without apparent progress: 0
step#: 94 learning rate: 0.1 average loss: 1.28144 steps without apparent progress: 21
step#: 202 learning rate: 0.1 average loss: 0.966533 steps without apparent progress: 7
step#: 313 learning rate: 0.1 average loss: 0.716872 steps without apparent progress: 41
step#: 424 learning rate: 0.1 average loss: 0.543538 steps without apparent progress: 45
step#: 534 learning rate: 0.1 average loss: 0.446105 steps without apparent progress: 58
step#: 642 learning rate: 0.1 average loss: 0.374214 steps without apparent progress: 51
step#: 739 learning rate: 0.1 average loss: 0.349595 steps without apparent progress: 0
step#: 831 learning rate: 0.1 average loss: 0.338897 steps without apparent progress: 111
step#: 923 learning rate: 0.1 average loss: 0.287758 steps without apparent progress: 74
step#: 1020 learning rate: 0.1 average loss: 0.281189 steps without apparent progress: 43
step#: 1118 learning rate: 0.1 average loss: 0.253426 steps without apparent progress: 179
step#: 1216 learning rate: 0.1 average loss: 0.242561 steps without apparent progress: 89
step#: 1314 learning rate: 0.1 average loss: 0.208625 steps without apparent progress: 64
step#: 1412 learning rate: 0.1 average loss: 0.20805 steps without apparent progress: 38
step#: 1510 learning rate: 0.1 average loss: 0.201305 steps without apparent progress: 173
step#: 1608 learning rate: 0.1 average loss: 0.200224 steps without apparent progress: 237
step#: 1706 learning rate: 0.1 average loss: 0.182268 steps without apparent progress: 30
step#: 1804 learning rate: 0.1 average loss: 0.169718 steps without apparent progress: 87
step#: 1902 learning rate: 0.1 average loss: 0.169856 steps without apparent progress: 82
step#: 2000 learning rate: 0.1 average loss: 0.167455 steps without apparent progress: 178
step#: 2098 learning rate: 0.1 average loss: 0.161374 steps without apparent progress: 277
step#: 2196 learning rate: 0.1 average loss: 0.15462 steps without apparent progress: 362
step#: 2294 learning rate: 0.1 average loss: 0.146062 steps without apparent progress: 252
step#: 2392 learning rate: 0.1 average loss: 0.153217 steps without apparent progress: 124
step#: 2490 learning rate: 0.1 average loss: 0.132255 steps without apparent progress: 289
step#: 2588 learning rate: 0.1 average loss: 0.131524 steps without apparent progress: 122
step#: 2686 learning rate: 0.1 average loss: 0.126059 steps without apparent progress: 221
step#: 2783 learning rate: 0.1 average loss: 0.126059 steps without apparent progress: 301
```

5 分钟后，2 级分的汽车轮毂被训练完成

将 汽车轮毂.svm_dnn_od 导出来使用即可



运行 Demo 测试模型

已在 zAI\Demo\ResNetImgClassifier2 中包含了汽车轮毂的检测 Demo

结合本文要点。运行 Demo，请多动脑筋去理解 [万物识别](#) 含义和它的制作工艺。

[模式识别->特诊识别](#) 有非常广阔应用前景，务必关心一下它的制作工艺，无论项目承接还是智能化体验，它的商业价值以及技术含量都领先于人脸。

2019-3

By qq600585