

.NET Conf China
2022



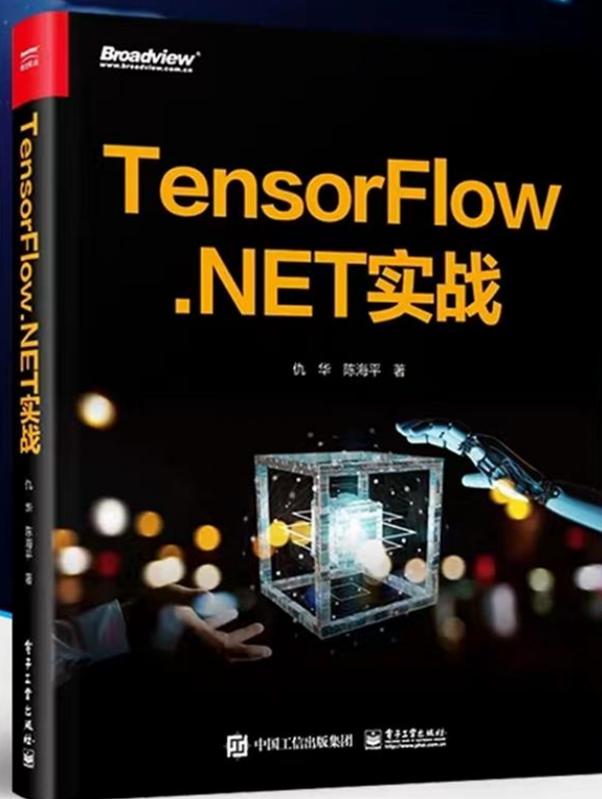
TensorFlow.NET实战

仇华
TCL - 资深视觉开发工程师



.NET Conf China

自我介绍



TensorFlow .NET 实战

仇华 陈海平 著

· 作者简介 ·



仇华

从事机器视觉和机器学习开发的工作14年。目前在TCL担任资深软件工程师，深耕 TensorFlow 苏州社区建设，获得谷歌深度学习（机器学习&机器视觉）证书和苏州市高级视觉工程师证书。

· 本书特色 ·

快速入门

本书介绍了核心API 的用法和基础示例，包括数据类型、张量、Eager Mode、自动求导、线性回归、逻辑回归、tf.data、深度神经网络和AutoGraph 机制，读者可以通过学习快速入门。

快速应用

本书演示了.NET Keras 的用法，包括模型、网络层、常用API、模型搭建和模型训练，读者可以由此快速掌握主流的深度学习方法。

快速落地

本书有大量的生产应用和案例实操，包括GPU 环境搭建、自定义数据集训练、图像分类、目标检测、迁移学习、自然语言处理、生成对抗网络和F#应用案例，每个案例均有完整的代码，帮助开发者快速在实际项目中开展AI技术落地。



演讲内容

- 智能制造中的AI视觉
- .NET深度学习解决方案
- TensorFlow.NET实战教程



.NET Conf China

智能制造中的AI视觉



疫情下制造企业的痛点

疫情放大了制造企业在自动化、信息化方面的短板，应对市场变化，**制造企业智能化** 转型迫在眉睫。



- 客户订单不均衡，市场急剧萎缩
- 急单、插单频繁发生，交货问题更加突出

客户订单问题



- 产品质量问题多、返工多、客户投诉多、物料损耗大、报废多
- 设备停机多、各种异常和等待，隐形浪费很大

企业生产问题



- 需要的物料无法及时到货，不需要的物料长时间呆滞；急着要发货的产品不能及时排产。

供应链问题



- 车间现场半成品多、不良品多，堆放混乱、管理困难
- 跨部门、跨企业协同难，沟通冲突大、决策难。

企业管理问题



- 疫情导致招人难，留人难，员工流失率居高不下

企业用人问题

制造企业危机下 应对策略：规划好企业的**智能制造** 路线, 提前做好风险预防, 增强企业的生存能力及**综合竞争力**

什么是智能制造



智能制造（Intelligent Manufacturing, IM）是一种由智能机器和人类专家共同组成的人机一体化智能系统，它在制造过程中能进行智能活动，诸如分析、推理、判断、构思和决策等。通过人与智能机器的合作共事，去扩大、延伸和部分地取代人类专家在制造过程中的脑力劳动。它把制造自动化的概念更新，扩展到**柔性化、智能化和高度集成化**。



智能制造 政策解读

《“十四五”智能制造发展规划》中解读智能制造

到2025年 规模以上制造业企业大部分实现数字化网络化，重点行业骨干企业初步应用智能化。

到2035年 规模以上制造业企业全面普及数字化网络化，重点行业骨干企业基本实现智能化。

2025年的主要目标

——转型升级成效显著。**70%**的规模以上制造业企业基本实现数字化网络化，建成**500个**以上引领行业发展的智能制造示范工厂。制造业企业生产效率、产品良品率、能源资源利用率等显著提升，智能制造能力成熟度水平明显提升。

——供给能力明显增强。智能制造装备和工业软件技术水平和市场竞争力显著提升，市场满足率分别超过**70%**和**50%**。培育**150家**以上专业水平高、服务能力强的智能制造系统解决方案供应商。

中华人民共和国中央人民政府
www.gov.cn

国务院 总理 新闻 政策 互动

首页 > 政策 > 国务院政策文件库 > 国务院部门文件

标题：八部门关于印发《“十四五”智能制造发展规划》的通知

发文字号：工信部联规〔2021〕207号

主题分类：工业、交通\其他

成文日期：2021年12月21日

“六大行动”

智能制造技术攻关行动



智能制造示范工厂建设行动



行业智能化改造升级行动



智能制造装备创新发展行动



工业软件突破提升行动



智能制造标准领航行动



智能制造 关键举措



精益化

自动化

数字化

智能化

■ **智能制造目标：**综合运用精益、自动化、数字化和智能化等手段，提升企业制造能力、推动企业制造模式创新
(传统大规模量产的生产模式 → 柔性制造、绿色制造、分型制造)

实施策略

实施领域

精简重复操作型人员
随业务拓展扩大核心技术人员规模

检测

检测人力占比40%



传统视觉算法性能不足



品质追溯效率低

物流

物流人力占比15%



应用场景灵活性差



配置冗余影响效率

组装

线上组装人力占比10%



组装技术柔性应用难



型号切换效率低



精益 20%



自动化 40%

智能装备强相关



数字化 10%



智能化 30%

机器视觉 智能制造大环境的选择



机器视觉被称为智能制造的“智慧之眼”，为智能制造打开了新的“视”界。机器视觉是生产过程中数据采集的首选技术之一，是实现工业自动化和智能化的必要手段，相当于人类视觉在机器上的延伸。为了保持我国在世界制造业中的竞争地位，我国提出了三步走实现制造强国目标的《中国制造2025》规划纲要：第一步，到2025年迈入制造强国行列；第二步，到2035年我国制造业整体达到世界制造强国阵营中等水平；第三步，到新中国成立一百年时，我制造业大国地位更加巩固，综合实力进入世界制造强国前列

《中国制造2025》三步走战略



第一步

- **时间节点：**2025年
- **主要目标：**迈入制造强国行列



第二步

- **时间节点：**2035年
- **主要目标：**整体达到世界制造强国阵营中等水平



第三步

- **时间节点：**新中国成立百年
2049
- **主要目标：**迈入世界制造强国前列



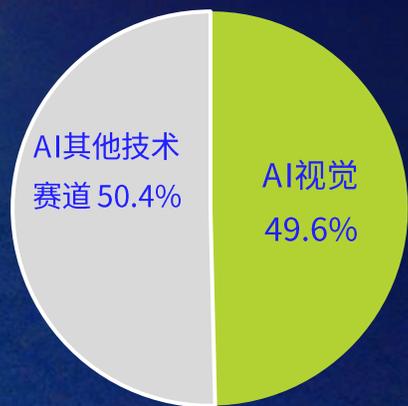
AI视觉商业化落地进程



千亿级大赛道初露端倪，成为人工智能产业规模的主战场

通过对下游行业需求统计测算，2021年我国AI视觉产品的市场规模占整个人工智能行业的49.6%，达到990亿元。和AI视觉有关的计算机通信设备销售、医疗器械等专用设备销售、工程建设、传统业务效益转化等带动相关产业规模超过3079亿元。从市场规模、场景泛用、带动作用来说，AI视觉领域已成为人工智能产业规模的主战场。AI视觉承接海量下游需求，未来增量动力依然强劲。

2021-2026年中国AI视觉核心产品及带动相关产业规模



2021年中国AI视觉市场在AI整体中占比**49.6%**

2021-2026年AI视觉**核心产品** CAGR **17.4%**

2021-2026年AI视觉**带动相关产业** CAGR **16.9%**



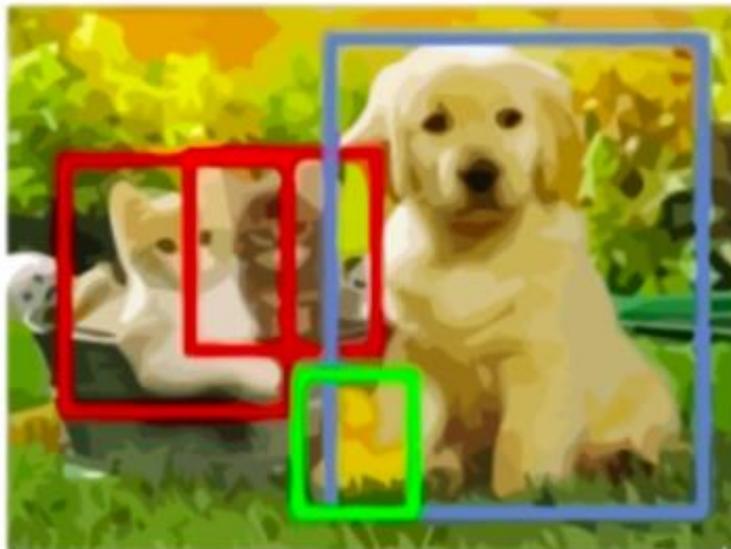
■ 中国计算机视觉核心产品规模 (亿元)

■ 中国计算机视觉带动相关产业规模 (亿元)

制造领域 深度学习CV应用



图像分类



目标检测



图像分割
(语义分割和实例分割)

GAN:

超分辨率

Super-Resolution

图像生成

Deep Dream

无监督学习:

异常检测



实际应用的 深度学习CV 制造现场



Data Augmentation
(数据增强)

设备系统集成

配套标注工具

交互UI

.NET 项目集成

传统算法协作开发



工业视觉环境差异



工业视觉 vs 传统图像 场景的差异?



制造环境 vs 非制造环境



图像

环境

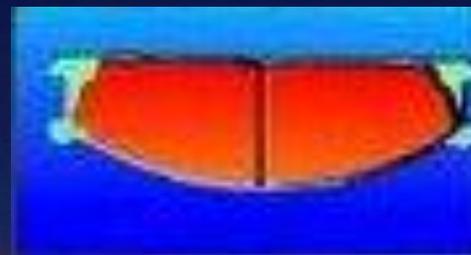
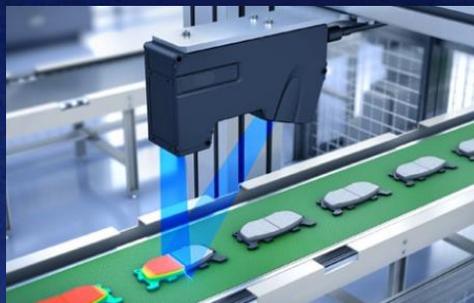
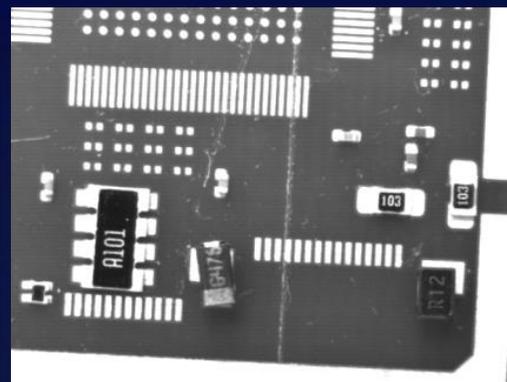
软件算法

	① 图像采集	② 成像环境	③ 负样本数量	④ 系统环境	⑤ 编程环境	⑥ 模型需求	⑦ 开发量	⑧ 算法特点
工业环境	种类多样 灰度图为主	稳定、单一	严重不足	系统老旧 、配置低	.NET为主	稳定高效	配套工具为主	传统算法为主 深度学习辅助
非工业环境	种类单一 彩色图为主	动态、复杂	可以满足	新、云系统 、配置合理	Python为主	不断更新迭代	模型开发优化	深度学习为主

1. 图像采集对比 - 制造业



制造业采集



1. 图像采集对比 – 非制造业



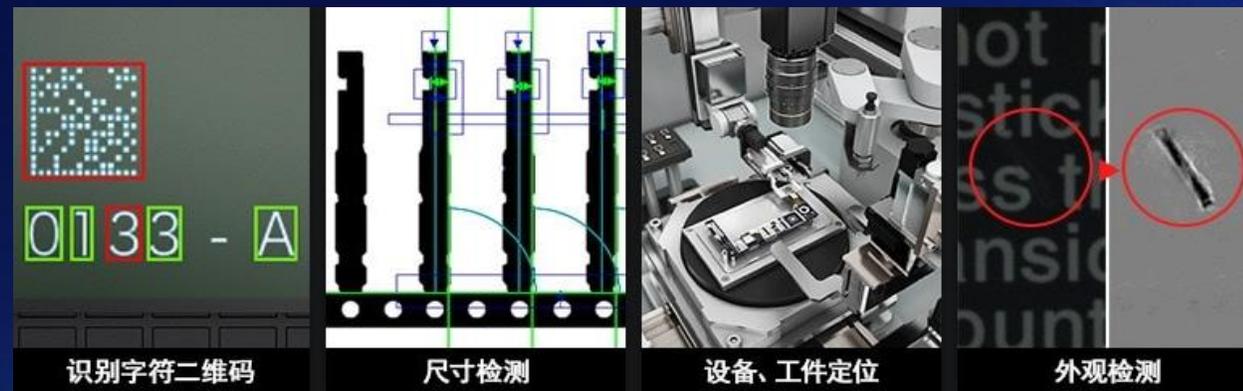
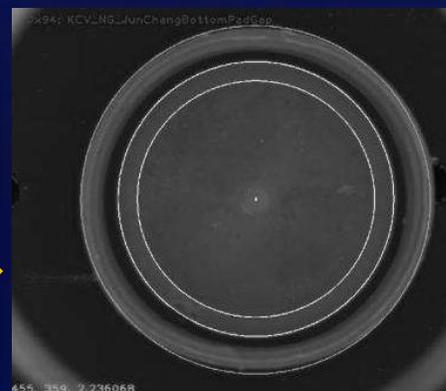
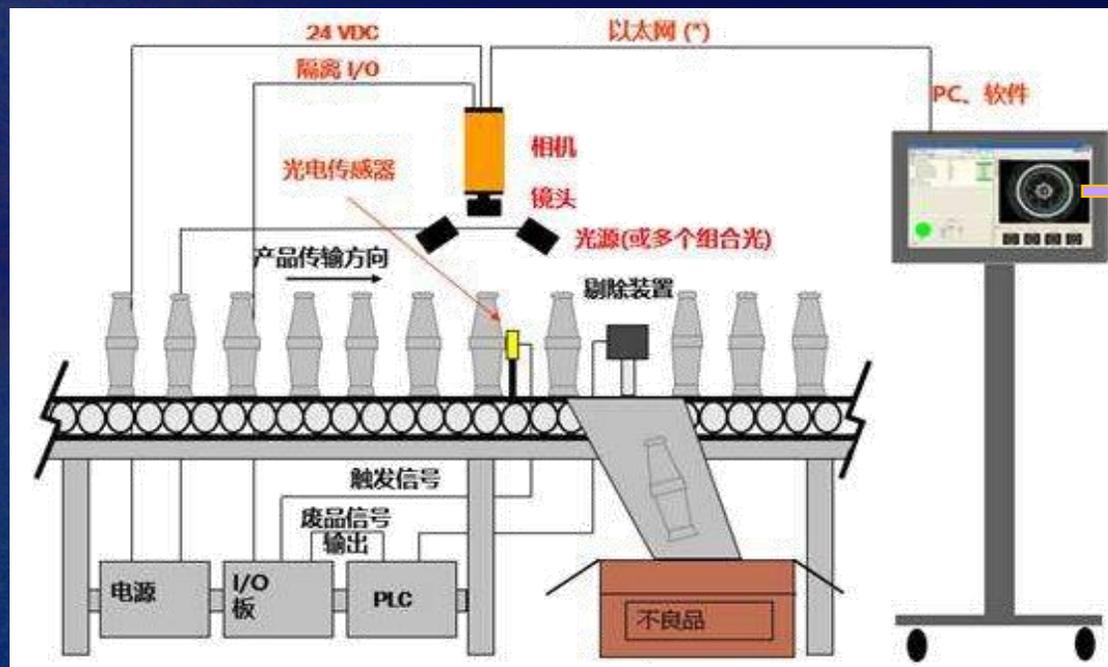
非制造业采集【模型一般基于非制造业图像适配】



2. 成像环境对比 – 制造业



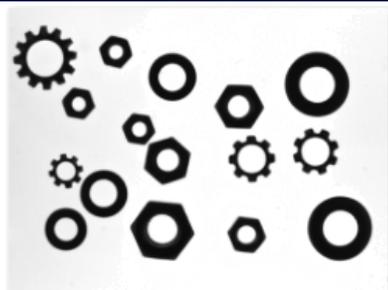
制造业图像



2. 成像环境对比 – 制造业图像集



mixed_01.png



mixed_02.png



mixed_03.png



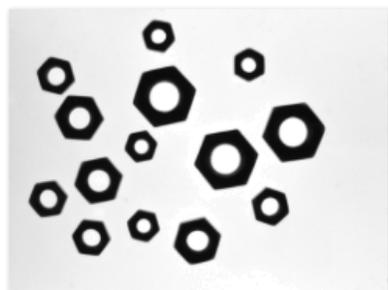
mixed_04.png



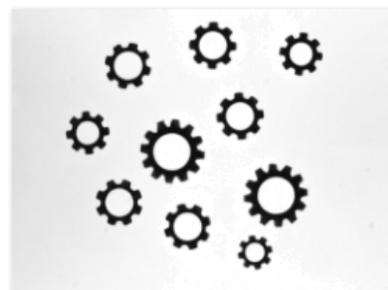
nuts_01.png



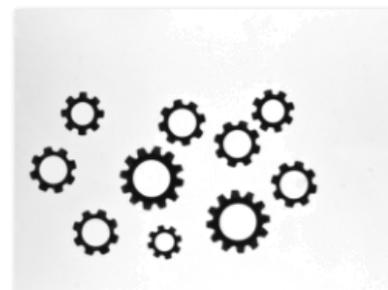
nuts_02.png



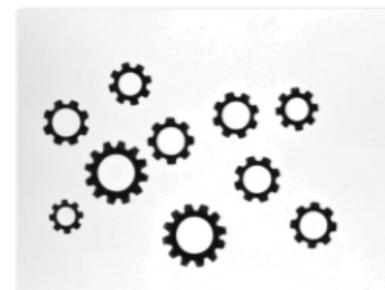
nuts_03.png



retainers_01.png



retainers_02.png



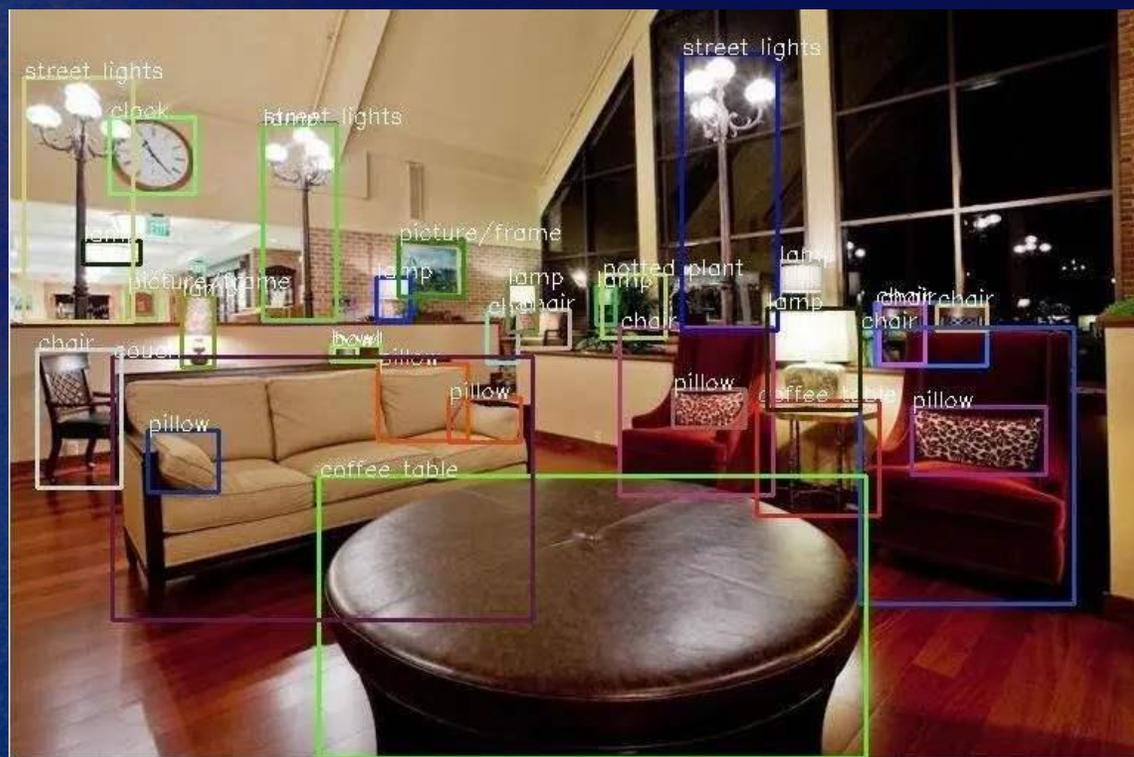
retainers_03.png



2. 成像环境对比 – 非制造业



非制造业图像



车牌图片



车牌号码



通道	车牌号码	颜色	类型	高度	识别时间	可信
1	京LC8521	蓝	普通车	101	88 毫秒	85
1	京QVM351	蓝	普通车	260	501 毫秒	75
1	京FD9862	蓝	普通蓝牌	139	33 毫秒	82
1	川RFD986	蓝	普通蓝牌	186	100 毫秒	63
1	冀G550QZ	蓝	普通蓝牌	520	197 毫秒	82
1	京FD9862	蓝	普通蓝牌	202	61 毫秒	84

3. 负样本严重不足



良品率 95% ↑

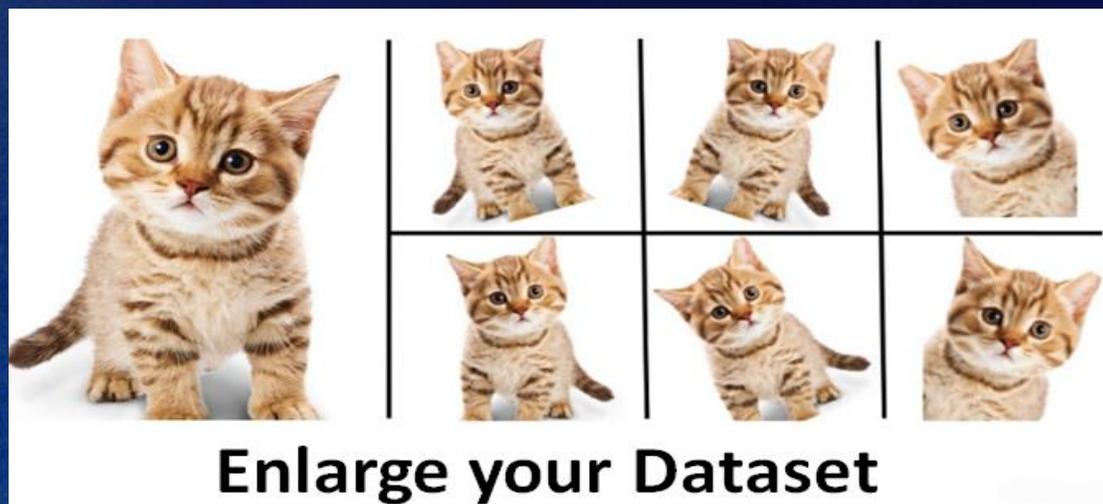


3. 数据增强方法



常规方式和 自主开发方式

	直接增加	直接复制，直接增加训练集比例
常规方式	几何变换类	翻转，旋转，裁剪，变形，缩放等
	颜色变换类	噪声、模糊、颜色变换、擦除、填充等
自主开发	随机采集	大图中随机采集特征区域
	图像融合	负样本目标特征区域 主动融合背景生成新图



4. 制造业系统环境



当前生产环境的工业软件主要有以下特点：

1. 封闭性和保密性，数据敏感，无法接入外部互联网；
2. 整体软件基础架构自主开发；
3. 运行速度和稳定性要求高，离线单机设备多；

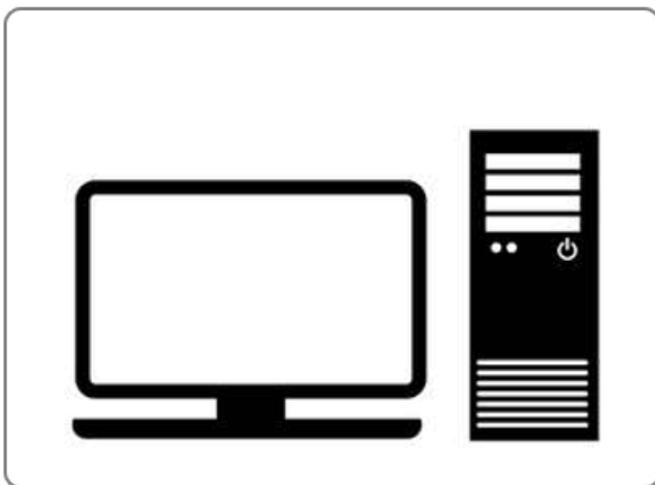


4. 老旧系统适配



POST通讯交互

客户端 [.NET]



功能模块:

- 图像获取
- 图像预处理
- 结果反馈至生产环境

Request
Json: Image

HTTP 协议: POST

Json: Result
Response

服务器 [Python]

* 也可以直接同1台PC同时做服务器和客户端, 127.0.0.1通讯



功能模块:

- DL模型训练
- DL模型预测
- 接收图像 / 反馈结果



5. 制造业视觉编程环境

TIOBE Index for November 2022



Nov 2022	Nov 2021	Change	Programming Language	Ratings	Change
1	1		 Python	17.18%	+5.41%
2	2		 C	15.08%	+4.35%
3	3		 Java	11.98%	+1.26%
4	4		 C++	10.75%	+2.46%
5	5		 C#	4.25%	-1.81%
6	6		 Visual Basic	4.11%	-1.61%
7	7		 JavaScript	2.74%	+0.08%
8	8		 Assembly language	2.18%	-0.34%
9	9		 SQL	1.82%	-0.30%
10	10		 PHP	1.69%	-0.12%

在深度学习的科研和互联网领域中，目前是以 Python 语言为主流，但是在传统的工业生产环境中，依然是 微软.NET 的天下，其中主要的开发语言为 C#。

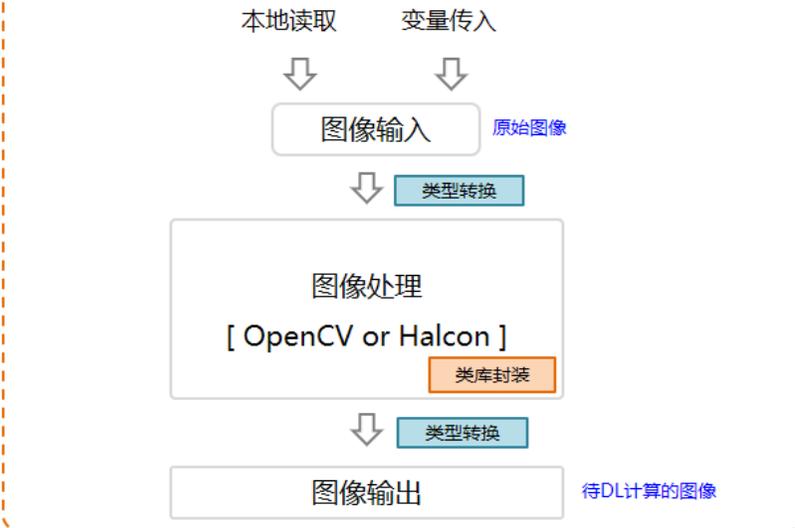


5. 制造业环境 部署深度学习 1#



主流流程：图像预处理

.NET



原理简述

训练：Python GPU版本 TensorFlow

推理：Python CPU or GPU版本 TensorFlow

模型部署：Python 加载模型，通过 Post 通讯，接收图像进行推理，返回 Json 格式结果

优缺点

优点：服务器/客户端分离，支持多客户端并行运算

缺点：需要安装和运行Python和.NET 2种框架，部署流程和架构复杂

Python

分支流程：训练

保存本地

GPU + TensorFlow
[训练]

类库封装

生成DL模型

Json: HTTP 协议

保存至变量

CPU or GPU + TensorFlow
[预测]

类库封装

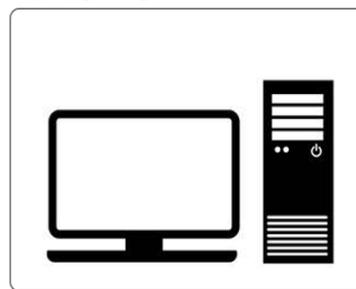
输出结果

Json: HTTP 协议

Python

分支流程：预测

客户端 [.NET]



功能模块：

- 图像获取
- 图像预处理
- 结果反馈至生产环境

Request

Json: Image

HTTP 协议: POST

Json: Result

Response

服务器 [Python]

* 也可以直接同1台PC同时做服务器和客户端，127.0.0.1通讯



RESTful 架构的 Web Service

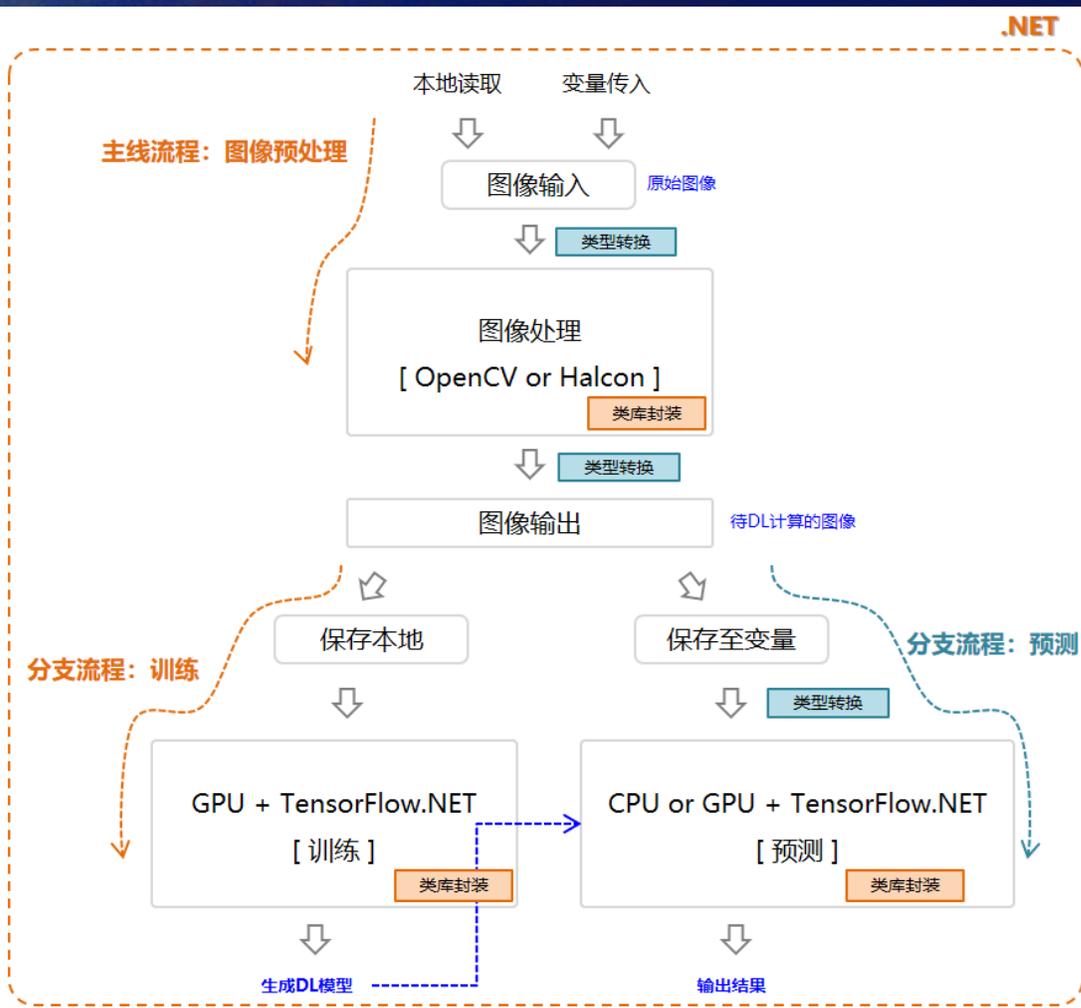
Apache + wsgi
+ Flask

功能模块：

- DL模型训练
- DL模型预测
- 接收图像 / 反馈结果

5. 制造业环境 部署深度学习 2#

推荐方式



原理简述

训练: GPU 版本 TensorFlow.NET

推理: CPU or GPU 版本 TensorFlow.NET

模型部署: TensorFlow.NET GPU版本 训练出的模型直接调用

优缺点

优点:

TensorFlow在.NET环境下的GPU支持

模型训练和部署 可以在同1套程序中集成, 无需外部通讯

Google官方推荐.NET开发者使用, 同时作为ML.NET的底层深度学习框架



Microsoft
.NET



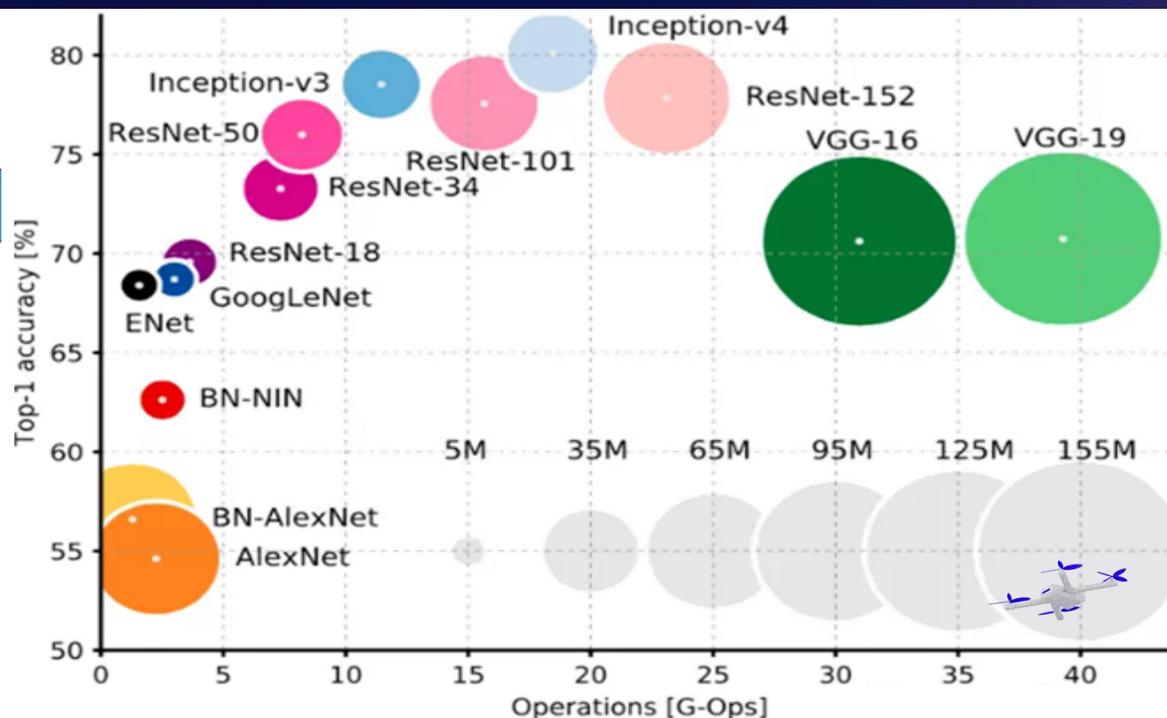
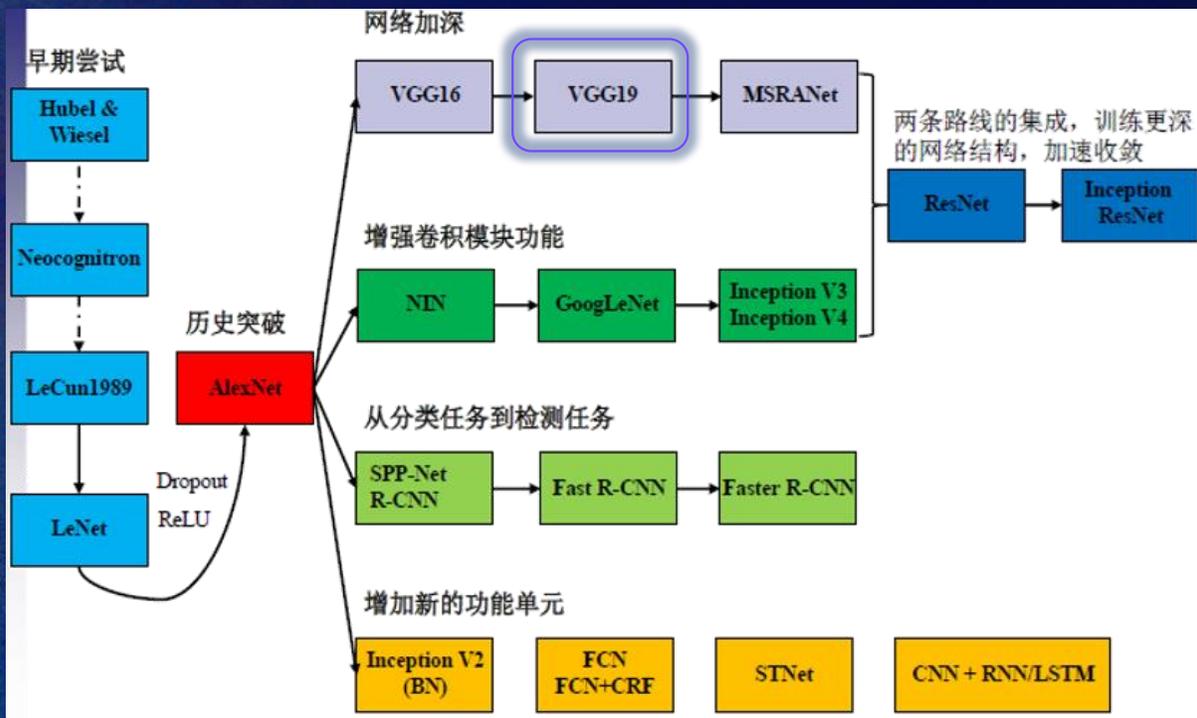
TensorFlow



6. 制造业视觉 网络模型特点



模型不需要前沿 需要稳定高效



7. 开发：算法落地的配套工具开发量占比高



The screenshot displays the Vision AI v1.1 interface. On the left, a terminal window shows the training progress of a CNN model, including loss and accuracy values over multiple iterations. The main area features a menu of tools (T001-T008) such as '特征小图手动提取', '小图标注分类工具', and '查看模型结构'. The bottom status bar provides system resource usage: CPU (49.9%), Mem (9.93 GB/31.94 GB), GPU (26%), GPU Mem (8125MiB/8192MiB), and Live/DetailLog options.

Left Panel (Terminal):

```

[09:14:44] CNN : iter 060: Loss=0.1929, Training Accuracy=94.00% 1195ms
[09:14:46] CNN : iter 070: Loss=0.2500, Training Accuracy=94.00% 1189ms
[09:14:47] CNN : iter 080: Loss=0.0915, Training Accuracy=98.00% 1176ms
[09:14:48] CNN : iter 090: Loss=0.2678, Training Accuracy=92.00% 1160ms
[09:14:49] CNN : iter 100: Loss=0.1509, Training Accuracy=92.00% 1183ms
[09:14:50] CNN : iter 110: Loss=0.2307, Training Accuracy=90.00% 1187ms
[09:14:52] CNN : iter 120: Loss=0.1304, Training Accuracy=94.00% 1327ms
[09:14:53] CNN : iter 130: Loss=0.1527, Training Accuracy=96.00% 1174ms
[09:14:54] CNN : iter 140: Loss=0.4369, Training Accuracy=94.00% 1193ms
[09:14:55] CNN : iter 150: Loss=0.1947, Training Accuracy=90.00% 1151ms
[09:14:56] CNN : iter 160: Loss=0.1102, Training Accuracy=98.00% 1205ms
[09:14:58] CNN : iter 170: Loss=0.2206, Training Accuracy=94.00% 1177ms
[09:14:59] CNN : iter 180: Loss=0.4649, Training Accuracy=94.00% 1190ms
[09:15:00] CNN : iter 190: Loss=0.3365, Training Accuracy=88.00% 1148ms
[09:15:00] CNN : -----
[09:15:00] CNN : gloabl steps: 1544, learning rate: 0.001, validation loss: 0.5617, validation accuracy: 83.64%
[09:15:00] CNN : -----
[09:15:02] CKPT Model is save.
[09:15:02] Training epoch: 9
[09:15:02] shuffle array list : 9663
[09:15:02] CNN : iter 000: Loss=0.2232, Training Accuracy=94.00% 2118ms
[09:15:03] CNN : iter 010: Loss=0.1761, Training Accuracy=94.00% 1202ms
[09:15:05] CNN : iter 020: Loss=0.1360, Training Accuracy=94.00% 1226ms
[09:15:06] CNN : iter 030: Loss=0.1132, Training Accuracy=96.00% 1162ms

```

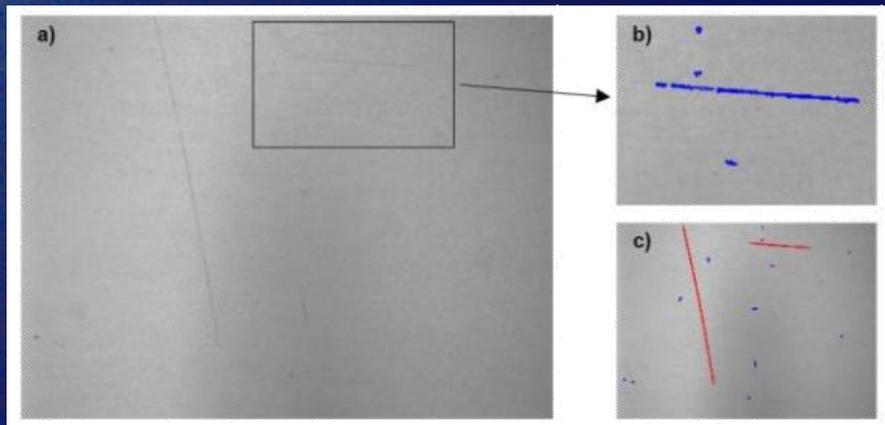
Main Panel (Tools):

- T001: 特征小图手动提取
- T002: 小图标注分类工具
- T003: 数据集增强算法
- T004: 文件批量操作
- T005: 查看模型结构
- T006: Netron 模型查看器
- T007: Mini Web
- T008: 负样本生成器

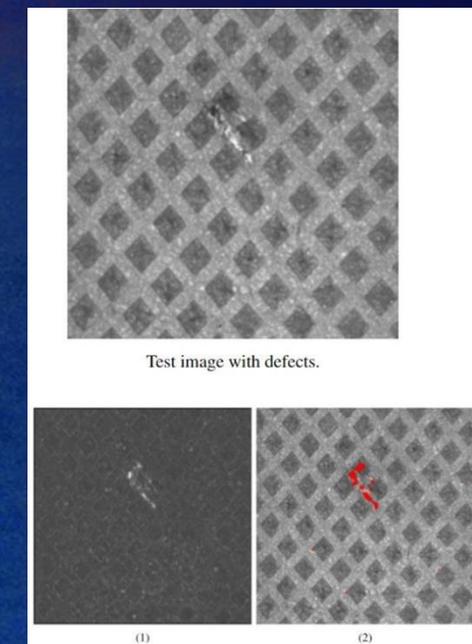
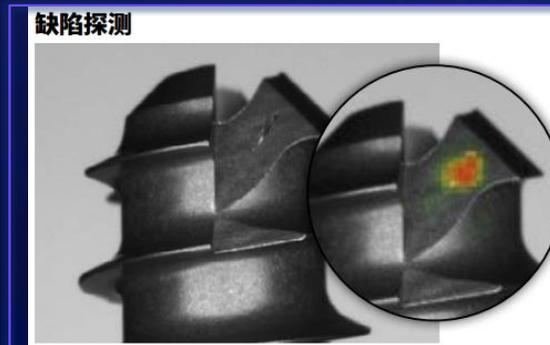
System Status (Bottom):

CPU:	100%	Mem:	100%	GPU:	100%	GPU Mem:	100%	Live	DetailLog
CPU:	49.9%	Mem:	9.93 GB/31.94 GB	GPU:	26%	GPU Mem:	8125MiB/8192MiB	<input checked="" type="checkbox"/> Live	<input type="checkbox"/> DetailLog

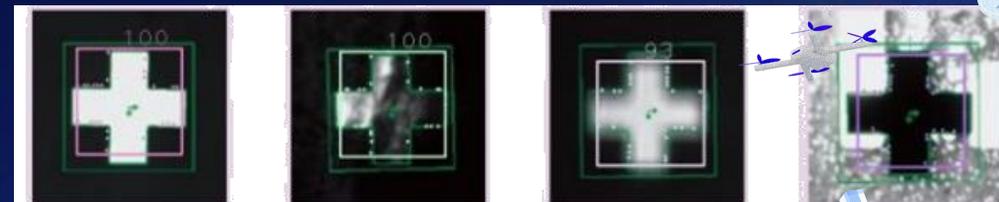
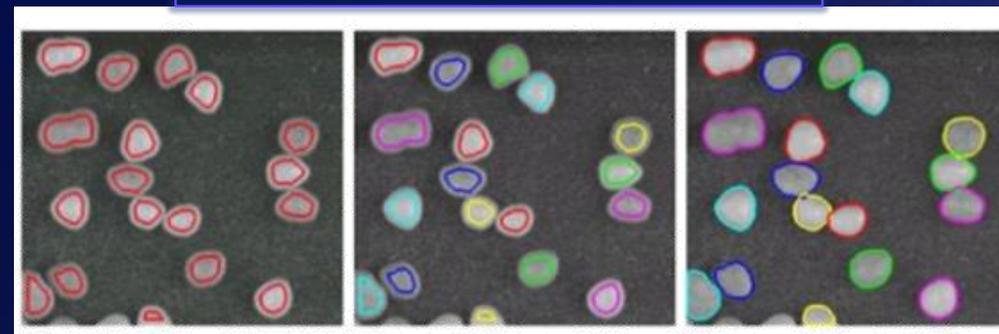
8. 算法特点：传统算法为主 深度学习辅助



Dynamic Threshold + Blob
or
DL Segmentation ?



Patch Feature Comparison
or
DL Object Detection ?



注册图像

缺损

轮廓模糊

灰度反转

.NET深度学习解决方案



```
Model model;
IEnumerable<ITrain, y_train, x_test, y_test>
Layers(layers: new LayersDef());
public void Main()
{
    //1. prepare data
    (x_train, y_train, x_test, y_test) = keras.datasets.mnist.load_data();
    x_train = x_train.reshape(6000, 784) / 255f;
    x_test = x_test.reshape(1000, 784) / 255f;

    //2. build model
    var inputs = keras.Input(shape: 784); // input layer
    var outputs = layers.Dense(64, activation: keras.activations.relu).Apply(inputs); // 1st dense layer
    outputs = layers.Dense(64, activation: keras.activations.relu).Apply(outputs); // 2nd dense layer
    outputs = layers.Dense(10).Apply(outputs); // output layer
    model = keras.Model(inputs, outputs, name: "mnist_model"); // build keras model
    model.summary(); // view model summary
    model.compile(loss: keras.losses.sparseCategoricalCrossentropy(from_logits: true),
        optimizer: keras.optimizers.Adam());
    metrics.Add("accuracy"); // compile keras model into tensorflow's static graph

    //3. train model by feeding data and labels
    model.fit(x_train, y_train, batch_size: 64, epochs: 3, validation_split: 0.2f);

    //4. evaluate the model
    model.evaluate(x_test, y_test, verbose: 2);

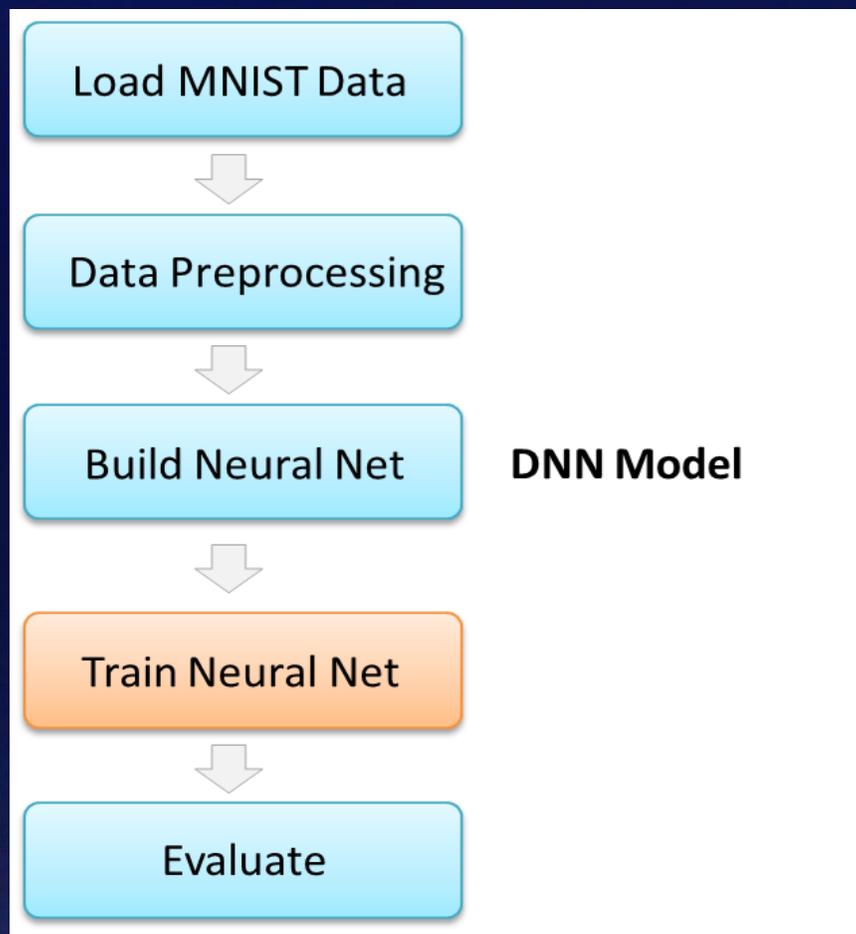
    //5. save and serialize model
    model.save("mnist_model");

    // reload the exact same model purely from the file:
    // model = keras.models.load_model("mnist_to_py_model");
}
```



深度学习 视觉开发方案

经典案例：MNIST 数据集采用 TensorFlow 2.x 的 Keras 模式下的 DNN 网络

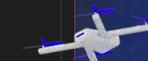


深度学习 视觉开发方案



TensorFlow.NET的C#代码实现

```
code sample.cs X
D: > LiveEvents > dotNetConf > 2022 > code sample.cs
1 // 输入层
2 var inputs = keras.Input(shape: (28, 28, 1));
3 // 第1层: 卷积层
4 var outputs = layers.Conv2D(32, kernel_size: 5, activation: keras.activations.Relu).Apply(inputs);
5 // 第2层: 池化层
6 outputs = layers.MaxPooling2D(2, strides: 2).Apply(outputs);
7 // 第3层: 卷积层
8 outputs = layers.Conv2D(64, kernel_size: 3, activation: keras.activations.Relu).Apply(outputs);
9 // 第4层: 池化层
10 outputs = layers.MaxPooling2D(2, strides: 2).Apply(outputs);
11 // 第5层: 展平层
12 outputs = layers.Flatten().Apply(outputs);
13 // 第6层: 全连接层
14 outputs = layers.Dense(1024).Apply(outputs);
15 // 第7层: 随机丢弃层
16 outputs = layers.Dropout(rate: 0.5f).Apply(outputs);
17 // 输出层
18 outputs = layers.Dense(10).Apply(outputs);
19 // 搭建Keras网络模型
20 model = keras.Model(inputs, outputs, name: "mnist_model");
21 // 显示模型概况
22 model.summary();
23 // 将Keras模型编译成TensorFlow的静态图
24 model.compile(loss: keras.losses.SparseCategoricalCrossentropy (from_logits: true),
25               optimizer: keras.optimizers.Adam(learning_rate: 0.001f),
26               metrics: new[] { "accuracy" });
27 // 训练模型
28 model.fit(x_train, y_train, batch_size: 64, epochs: 2, validation_split: 0.2f);
29 // 评估模型
30 model.evaluate(x_test, y_test, verbose: 2);
```



深度学习 视觉开发方案

【TensorFlow.NET的C#运行结果】

Model: mnist_model

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 784)	0
dense (Dense)	(None, 64)	50240
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 10)	650

Total params: 55050

Trainable params: 55050

Non-trainable params: 0

Training...

epoch: 0, loss: 2.3052168, accuracy: 0.046875

epoch: 1, loss: 0.34531808, accuracy: 0.9035208

epoch: 2, loss: 0.25493768, accuracy: 0.9276875

Testing...

iterator: 1, loss: 0.24668744, accuracy: 0.929454

项目基于 .NET Standard 2.0标准框架，
同时适配 .NET Framework、.NET Core
及 **.NET 5.0 / 6.0 / .NET 7.0**



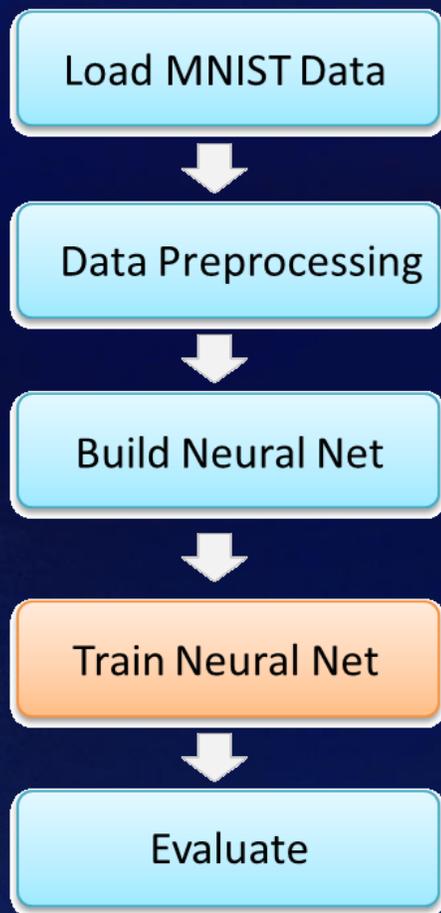
nuget 1 Million ↑



.NET 7



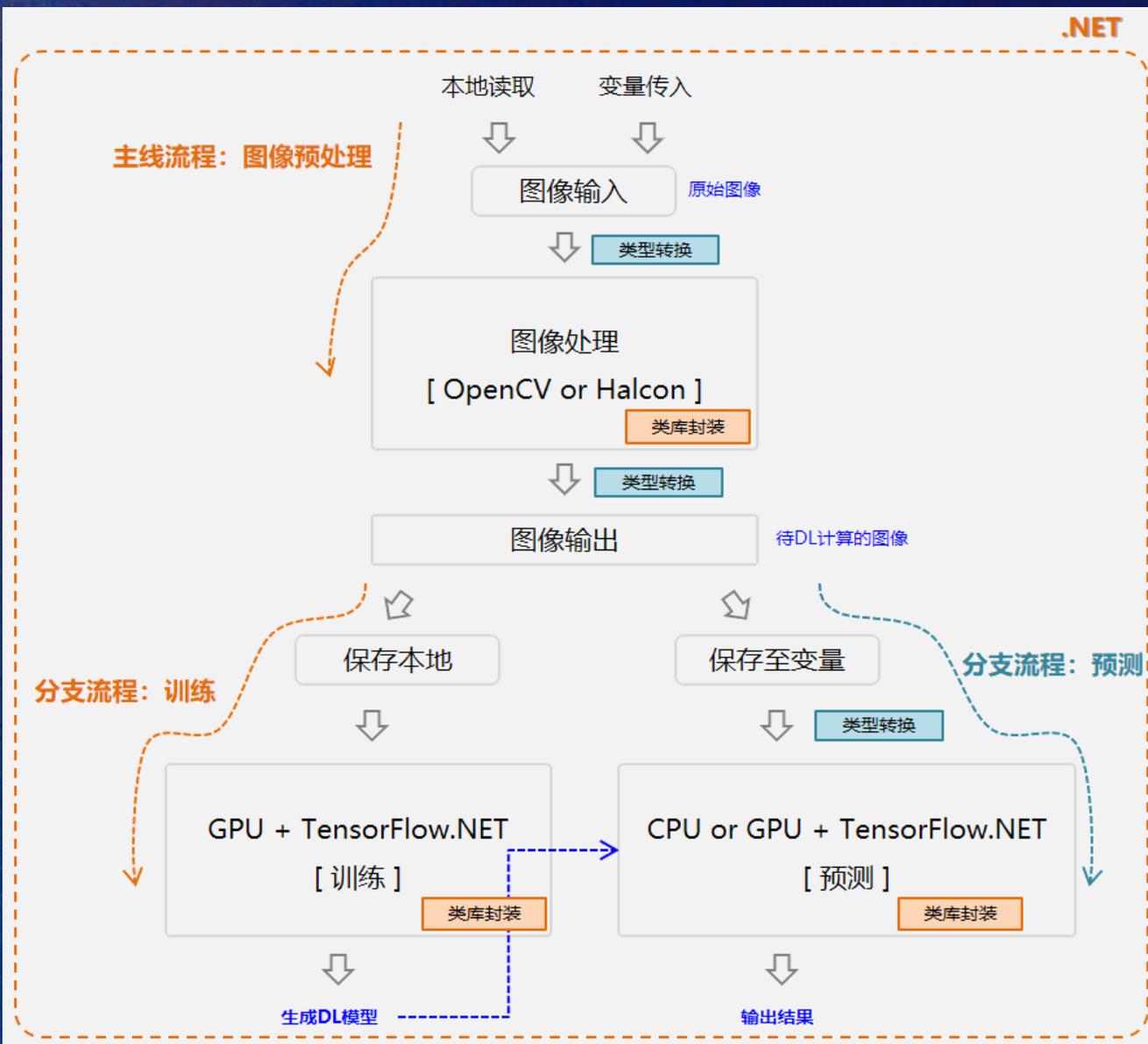
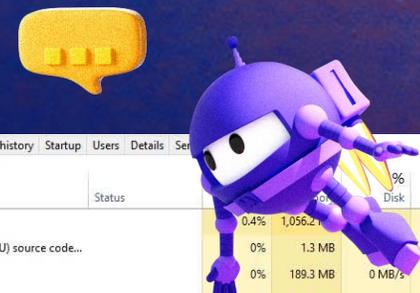
实际工业视觉的完整解决方案



主线流程



实际工业视觉的完整解决方案



.NET/ C# Binding VS Python Binding for TensorFlow 2.3 rc0

较Python性能大幅提升

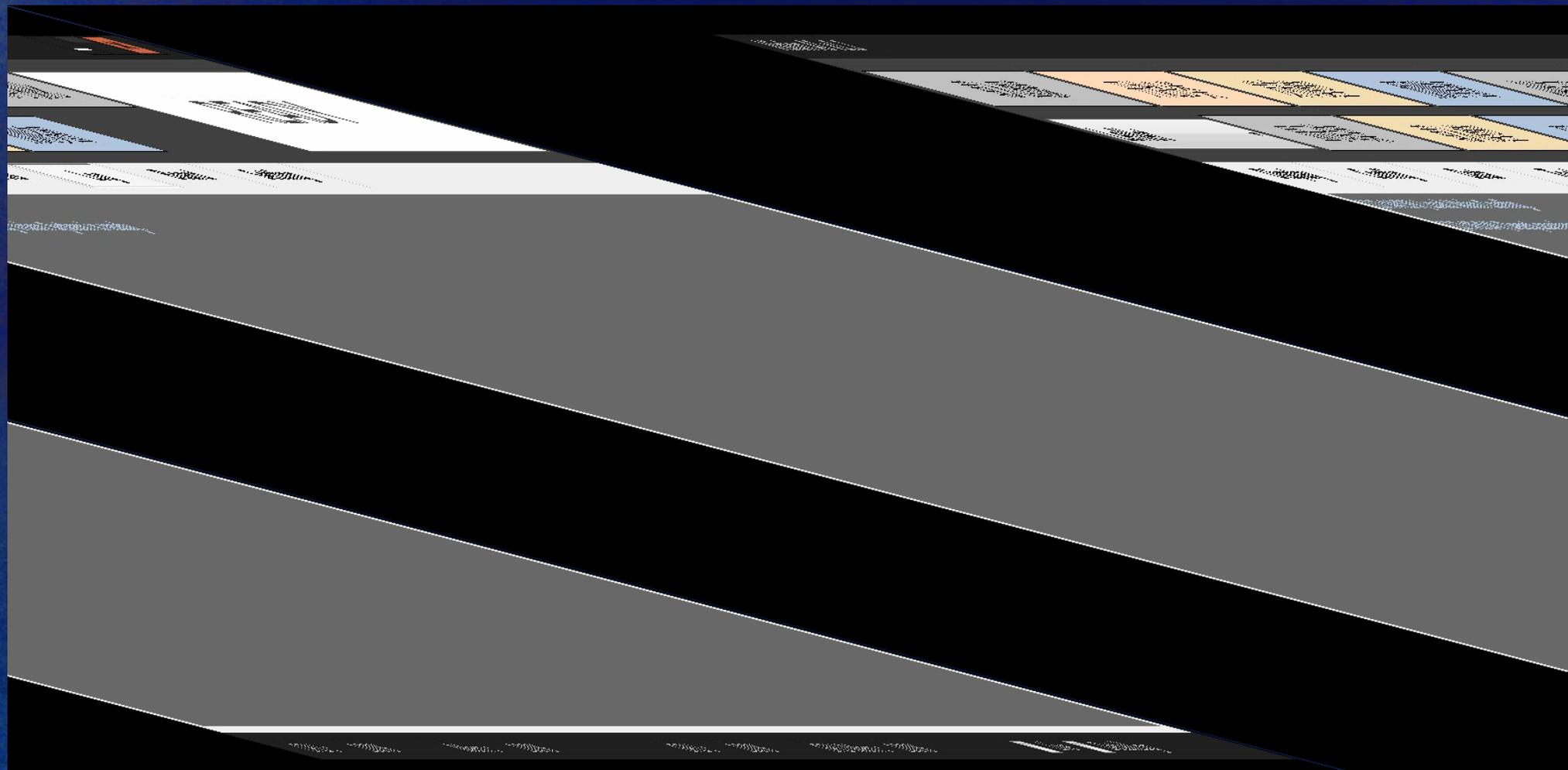
10,000 training steps for Linear Regression in Eager mode.
TensorFlow.NET is 2x faster than Python and uses 1/4 memory.

名称	修改日期	类型	大小
cublas64_100.dll	2019/1/24 23:31	应用程序扩展	65,734 KB
cuda64_100.dll	2019/1/24 23:31	应用程序扩展	407 KB
cuda64_100.dll	2019/1/24 23:31	应用程序扩展	1,034 KB
cufft64_100.dll	2019/1/24 23:31	应用程序扩展	99,650 KB
curand64_100.dll	2019/1/24 23:31	应用程序扩展	48,585 KB
cusolver64_100.dll	2019/1/24 23:31	应用程序扩展	125,563 KB
cusparse64_100.dll	2019/1/24 23:31	应用程序扩展	18 KB

一键部署GPU，最大化体现.NET优势，彻底解决GPU环境配置的繁琐问题，让你专注于深度学习算法和模型的开发。

GPU直接使用，无需部署环境

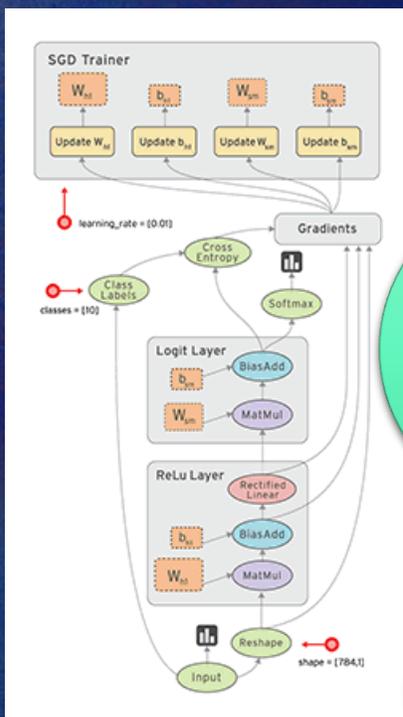
.NET 深度学习的工业视觉解决方案



TensorFlow.NET实战教程

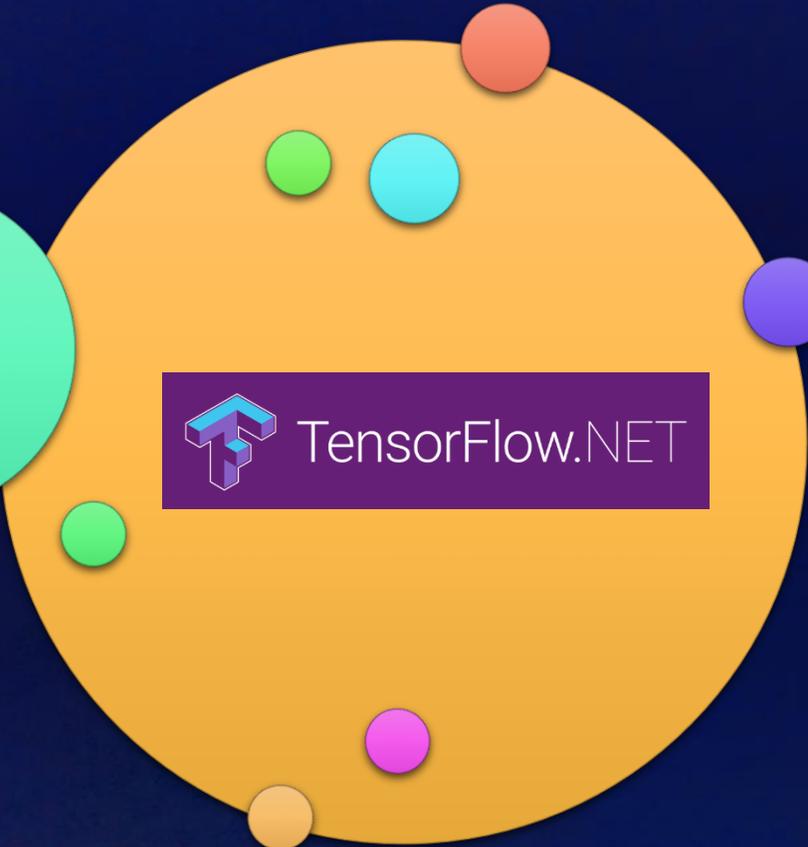


TensorFlow.NET 综述



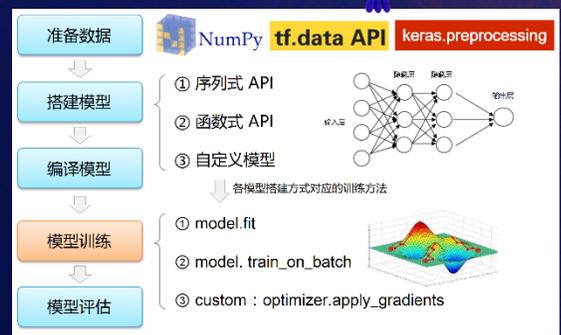
TensorFlow.NET API

Eager Mode



.NET Keras

.NET Keras API & Models



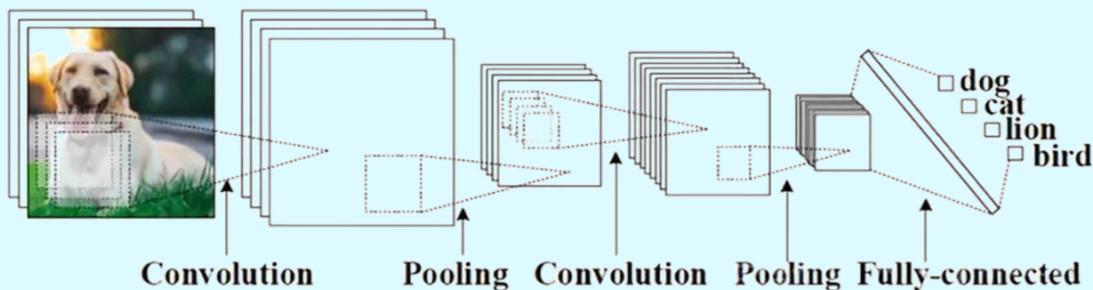
.NET Models

- SciSharp.Models.AudioRecognition
- SciSharp.Models.Core
- SciSharp.Models.ImageClassification
- SciSharp.Models.ObjectDetection
- SciSharp.Models.TextClassification
- SciSharp.Models.TimeSeries
- SciSharp.Models.Transformer

TensorFlow.NET 图像分类



CNN

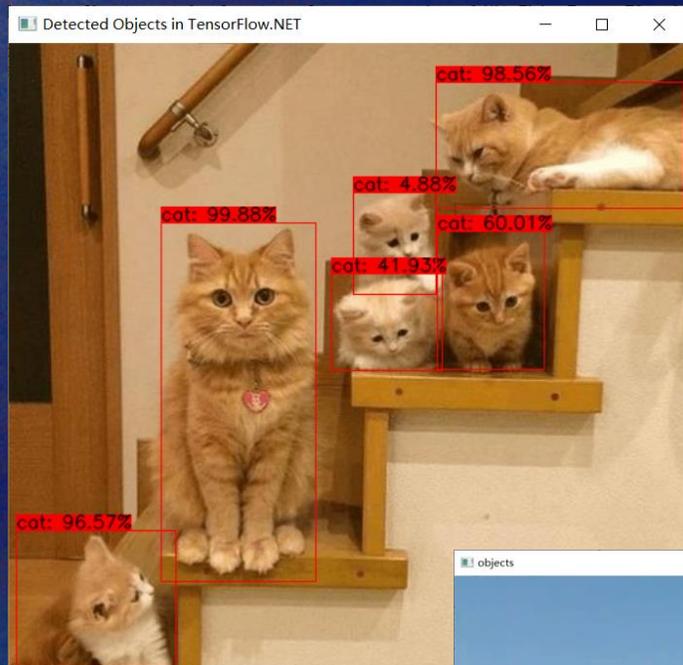


```
// 输入层
var inputs = keras.Input(shape: (28, 28, 1));
// 第1层: 卷积层
var outputs = layers.Conv2D(32, kernel_size: 5, activation: keras.activations.Relu).Apply(inputs);
// 第2层: 池化层
outputs = layers.MaxPooling2D(2, strides: 2).Apply(outputs);
// 第3层: 卷积层
outputs = layers.Conv2D(64, kernel_size: 3, activation: keras.activations.Relu).Apply(outputs);
// 第4层: 池化层
outputs = layers.MaxPooling2D(2, strides: 2).Apply(outputs);
// 第5层: 展平层
outputs = layers.Flatten().Apply(outputs);
// 第6层: 全连接层
outputs = layers.Dense(1024).Apply(outputs);
// 第7层: 随机丢弃层
outputs = layers.Dropout(rate: 0.5f).Apply(outputs);
// 输出层
outputs = layers.Dense(10).Apply(outputs);
// 搭建Keras网络模型
model = keras.Model(inputs, outputs, name: "mnist_model");
// 显示模型概况
model.summary();
// 将Keras模型编译成TensorFlow的静态图
model.compile(loss: keras.losses.SparseCategoricalCrossentropy (from_logits: true),
  optimizer: keras.optimizers.Adam(learning_rate: 0.001f),
  metrics: new[] { "accuracy" });

// 训练模型
// 使用输入数据和标签来训练模型
model.fit(x_train, y_train, batch_size: 64, epochs: 2, validation_split: 0.2f);
// 评估模型
model.evaluate(x_test, y_test, verbose: 2);
```



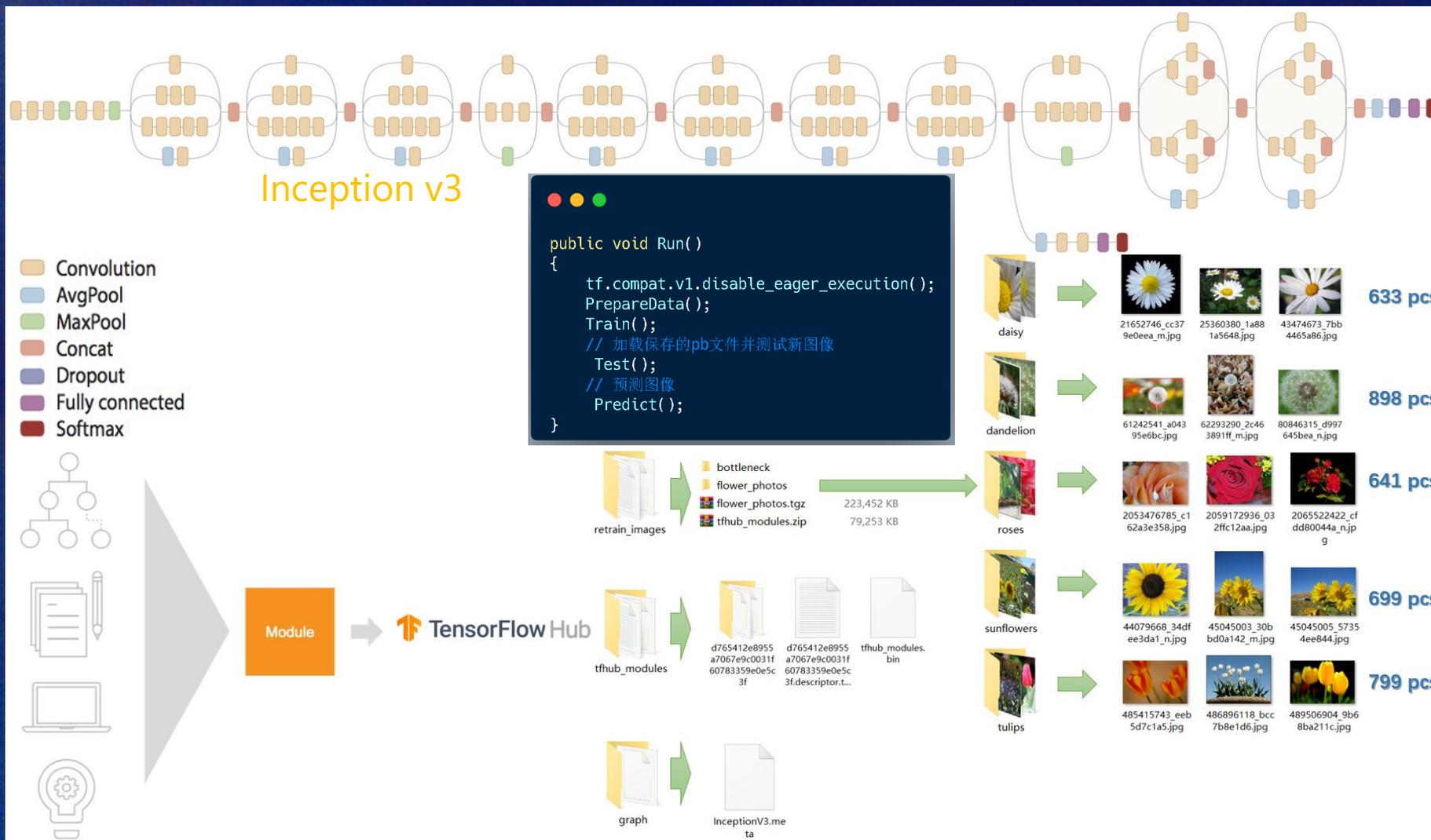
TensorFlow.NET 目标检测



```
var graph = ImportGraph();
using (var sess = new Session(graph))
{
    var original_image_raw = cv2.imread(AppDomain.CurrentDomain.BaseDirectory + @"yolov3\cat_face.jpg");
    var original_image = cv2.cvtColor(original_image_raw, ColorConversionCodes.COLOR_BGR2RGB);
    var original_image_size = (original_image.shape[0], original_image.shape[1]);
    var image_data = image_preporcess(original_image, (input_size, input_size));
    image_data = image_data[np.newaxis, Slice.Ellipsis];
    var (pred_sbbox, pred_mbbox, pred_lbbox) = sess.run ((return_tensors[1], return_tensors[2],
return_tensors[3]),(return_tensors[0], image_data));
    var pred_bbox = np.concatenate((np.reshape(pred_sbbox, (-1, 5 + num_classes)),np.reshape(pred_mbbox, (-1, 5
+ num_classes)), np.reshape
                                (pred_lbbox, (-1, 5 + num_classes))), axis: 0);
    var bboxes = postprocess_boxes(pred_bbox, original_image_size, input_size, 0.03f);//现况: 0.3f
    var bboxess = nms(bboxes, 0.3f, method: "nms");//现况: 0.5f
    var image = draw_bbox(original_image_raw, bboxess);
    cv2.imshow("Detected Objects in TensorFlow.NET", image);
}
```



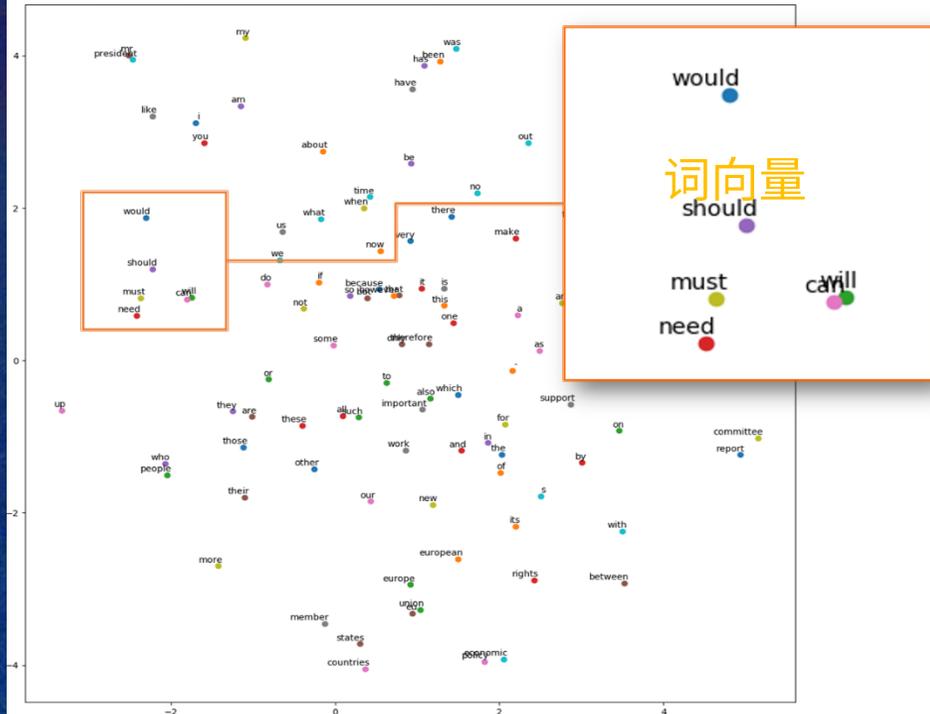
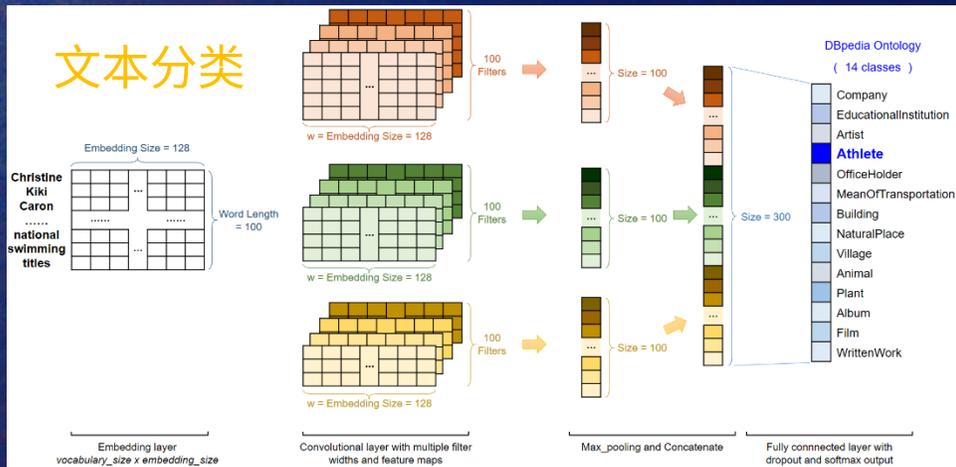
TensorFlow.NET 迁移学习



TensorFlow.NET 自然语言处理



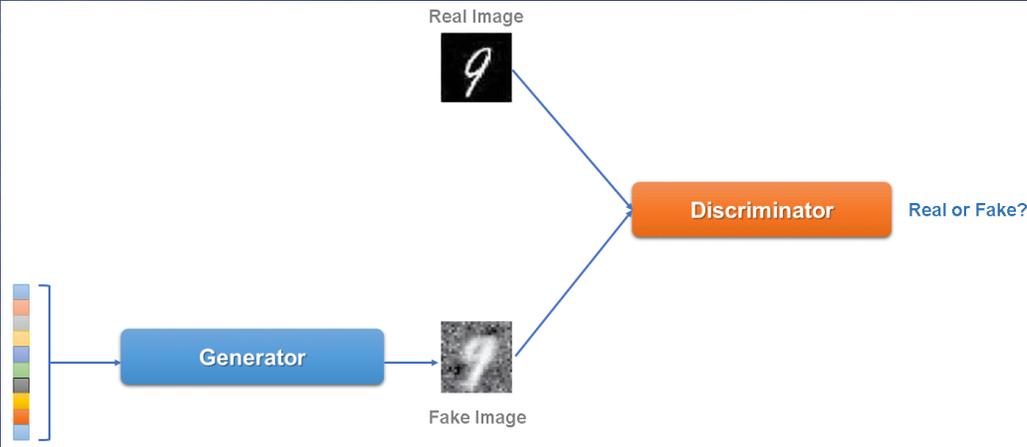
文本分类



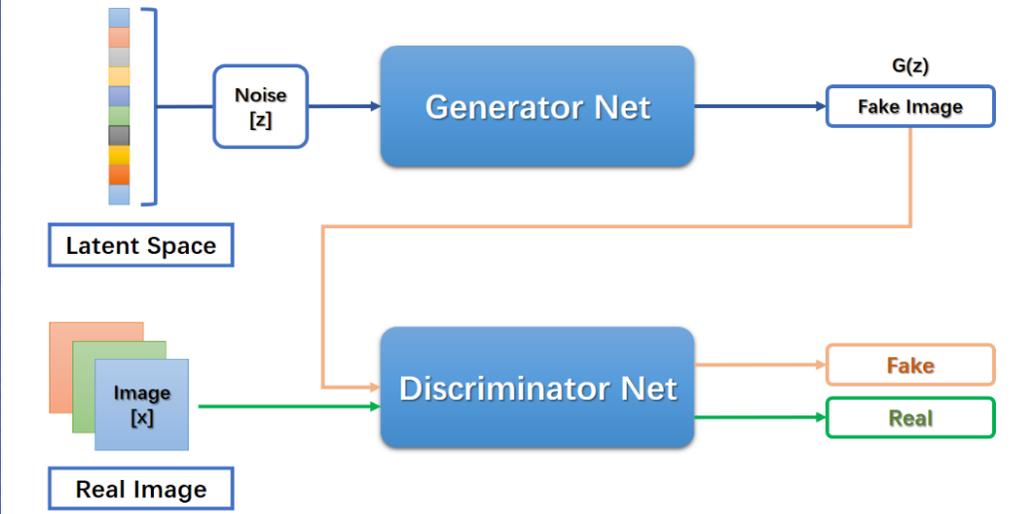
```
var embedding_size = 128;
var learning_rate = 0.001f;
var filter_sizes = new int[3, 4, 5];
var num_filters = 100;
var x = tf.placeholder(tf.int32, new TensorShape(-1, document_max_len), name: "x");
var y = tf.placeholder(tf.int32, new TensorShape(-1), name: "y");
var is_training = tf.placeholder(tf.@bool, new TensorShape(), name: "is_training");
var global_step = tf.Variable(0, trainable: false);
var keep_prob = tf.where(is_training, 0.5f, 1.0f);
Tensor x_emb = null;
tf_with(tf.name_scope("embedding"), scope =>
    { var init_embeddings = tf.random_uniform(new int[] { vocabulary_size, embedding_size });
      var embeddings = tf.compat.v1.get_variable("embeddings", initializer: init_embeddings);
      x_emb = tf.nn.embedding_lookup(embeddings, x);
      x_emb = tf.expand_dims(x_emb, -1); });
var pooled_outputs = new List<Tensor>();
for (int len = 0; len < filter_sizes.Rank; len++)
{ int filter_size = filter_sizes.GetLength(len);
  var conv = keras.layers.Conv2D(
    filters: num_filters,
    kernel_size: new int[] { filter_size, embedding_size },
    strides: new int[] { 1, 1 },
    padding: "VALID",
    activation: tf.nn.relu).Apply(x_emb);
  var pool = keras.layers.max_pooling2d( conv,
    pool_size: new[] { document_max_len - filter_size + 1, 1 },
    strides: new[] { 1, 1 },
    padding: "VALID");
  pooled_outputs.Add(pool); }
```



TensorFlow.NET 生成对抗网络



$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$



```
var model = keras.Sequential();
model.add(keras.layers.Dense(img_rows / 4 * img_cols / 4 * 256, activation: activation, input_shape: 100));
model.add(keras.layers.BatchNormalization(momentum: 0.8f));
model.add(keras.layers.LeakyReLU(LeakyReLU_alpha));
model.add(keras.layers.Reshape((7, 7, 256)));

model.add(keras.layers.UpSampling2D());
model.add(keras.layers.Conv2D(128, 3, 1, padding: "same", activation: activation));
model.add(keras.layers.BatchNormalization(momentum: 0.8f));
model.add(keras.layers.LeakyReLU(LeakyReLU_alpha));

model.add(keras.layers.UpSampling2D());
model.add(keras.layers.Conv2D(64, 3, 1, padding: "same", activation: activation));
model.add(keras.layers.BatchNormalization(momentum: 0.8f));
model.add(keras.layers.LeakyReLU(LeakyReLU_alpha));

model.add(keras.layers.Conv2D(32, 3, 1, padding: "same", activation: activation));
model.add(keras.layers.BatchNormalization(momentum: 0.8f));
model.add(keras.layers.LeakyReLU(LeakyReLU_alpha));

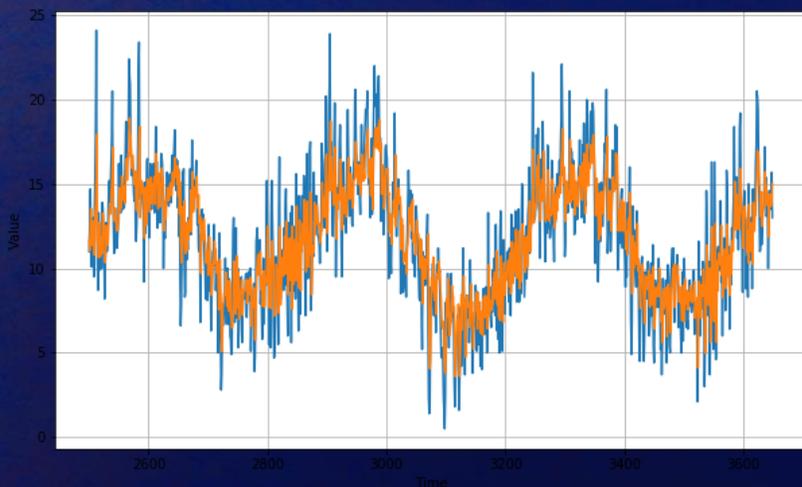
model.add(keras.layers.Conv2D(1, 3, 1, padding: "same", activation: "tanh"));
model.summary();
```

conv2d_3 (Conv2D)

(None, 28, 28, 1)



TensorFlow.NET 时序预测



LinearModel

```
public class LinearModel : ModelBase, ITimeSeriesTask
{
    protected override Model BuildModel()
    {
        var model = keras.Sequential(new List<ILayer> { keras.layers.Dense(units: 1) });

        /*early_stopping = keras.callbacks.EarlyStopping(monitor = "val_loss",
            patience = patience,
            mode = 'min')*/

        model.compile(loss: keras.losses.MeanSquaredError(),
            optimizer: keras.optimizers.Adam(),
            metrics: new[] { "mae" });

        return model;
    }
}
```

DenseModel

```
public class DenseModel : ModelBase, ITimeSeriesTask
{
    protected override Model BuildModel()
    {
        var model = keras.Sequential(new List<ILayer>
        {
            // Shape: (time, features) => (time*features)
            keras.layers.Flatten(),
            keras.layers.Dense(units: 32, activation: "relu"),
            keras.layers.Dense(units: 32, activation: "relu"),
            keras.layers.Dense(units: 1),
            // Add back the time dimension.
            // Shape: (outputs) => (1, outputs)
            keras.layers.Reshape((1, -1))
        });

        /*early_stopping = keras.callbacks.EarlyStopping(monitor = "val_loss",
            patience = patience,
            mode = 'min')*/

        model.compile(loss: keras.losses.MeanSquaredError(),
            optimizer: keras.optimizers.Adam(),
            metrics: new[] { "mae" });

        return model;
    }
}
```

ConvolutionalModel

```
public class ConvolutionalModel : ModelBase, ITimeSeriesTask
{
    protected override Model BuildModel()
    {
        var model = keras.Sequential(new List<ILayer>
        {
            keras.layers.Conv1D(filters: 32, kernel_size: _args.InputWidth, activation: "relu"),
            keras.layers.Dense(units: 32, activation: "relu"),
            keras.layers.Dense(units: 1)
        });

        /*early_stopping = keras.callbacks.EarlyStopping(monitor = "val_loss",
            patience = patience,
            mode = 'min')*/

        model.compile(loss: keras.losses.MeanSquaredError(),
            optimizer: keras.optimizers.Adam(),
            metrics: new[] { "mae" });

        return model;
    }
}
```



TensorFlow.NET with F#



F# empowers everyone to write succinct, robust and performant code



Don Syme

Basic Model

- Hello World [C#](#), [F#](#)
- Basic Operations [C#](#), [F#](#)
- Linear Regression in Graph mode [C#](#), [F#](#)
- Linear Regression in Eager mode [C#](#), [F#](#)
- Linear Regression in Keras [C#](#)
- Logistic Regression in Graph mode [C#](#), [F#](#)
- Logistic Regression in Eager mode [C#](#), [F#](#)
- Nearest Neighbor [C#](#), [F#](#)
- Naive Bayes Classification [C#](#), [F#](#)
- K-means Clustering [C#](#)

Neural Network

- Full Connected Neural Network in Eager mode [C#](#), [F#](#)
- Full Connected Neural Network (Keras) [C#](#), [F#](#)
- NN XOR [C#](#)
- Object Detection in MobileNet [C#](#)
- MNIST FNN in Keras Functional API [C#](#), [F#](#)
- MNIST CNN in Graph mode [C#](#), [F#](#)
- MNIST CNN in Eager mode [C#](#), [F#](#)
- MNIST CNN in Keras SubClass [C#](#), [F#](#)
- MNIST RNN [C#](#)
- MNIST LSTM [C#](#)
- Image Classification in Keras Sequential API [C#](#), [F#](#)
- Image Recognition Inception [C#](#), [F#](#)
- Toy ResNet in Keras Functional API [C#](#), [F#](#)
- Transfer Learning for Image Classification in InceptionV3 [C#](#)
- CNN In Your Own Dataset [C#](#), [F#](#)

Natural Language Processing

- Binary Text Classification [C#](#)
- CNN Text Classification [C#](#)
- Named Entity Recognition [C#](#)

Time Series

- Weather Prediction [C#](#)



Thank you!

Let's build amazing apps with .NET 7
get.dot.net/7

