高性能计算第四节

目标检测算法

本节主要内容

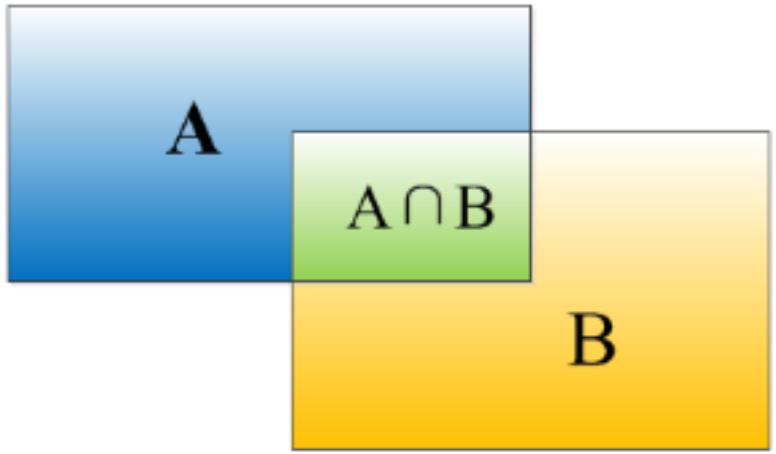
- R-CNN模型
- YOLO模型
- SSD模型
- TensorRT框架入门
- 练习

目标检测算法

- 目标检测是图像处理和计算机视觉学科的重要分支，也是智能监控系统的核心部分。
- 目标检测是泛身份识别领域的一个基础性的算法，对人脸识别、步态识别、人群计数、实例分割等任务起着至关重要的作用
- 基于深度学习的两种目标检测算法：One-Stage、Two-Stage。

重叠度（IOU）

IOU：定义了两个bounding box的重叠度：

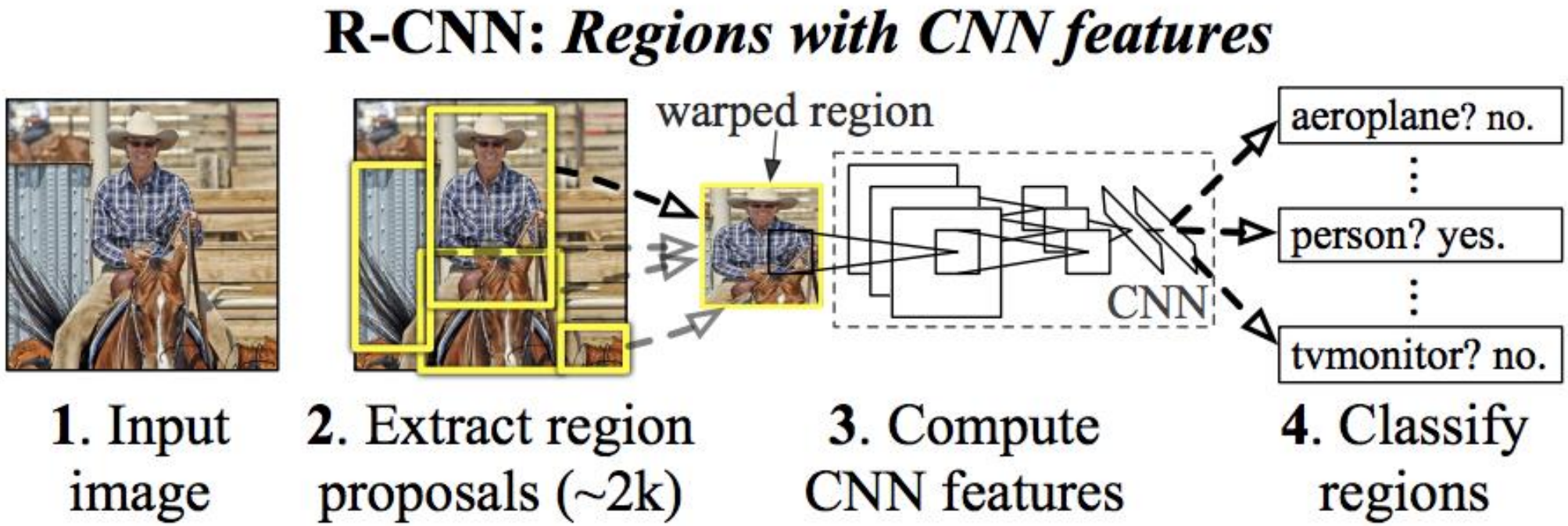



```
float overlap(float x1, float w1, float x2, float w2)
{
    float l1 = x1 - w1/2;
    float l2 = x2 - w2/2;
    float left = l1 > l2 ? l1 : l2;
    float r1 = x1 + w1/2;
    float r2 = x2 + w2/2;
    float right = r1 < r2 ? r1 : r2;
    return right - left;
}
  float box_intersection(box a, box b)
  {
      float w = overlap(a.x, a.w, b.x, b.w);
      float h = overlap(a.y, a.h, b.y, b.h);
      if(w < 0 || h < 0) return 0;
      float area = w*h;
      return area;
  }

  float box_union(box a, box b)
  {
      float i = box_intersection(a, b);
      float u = a.w*a.h + b.w*b.h - i;
      return u;
  }
```

NMS（非极大值抑制）

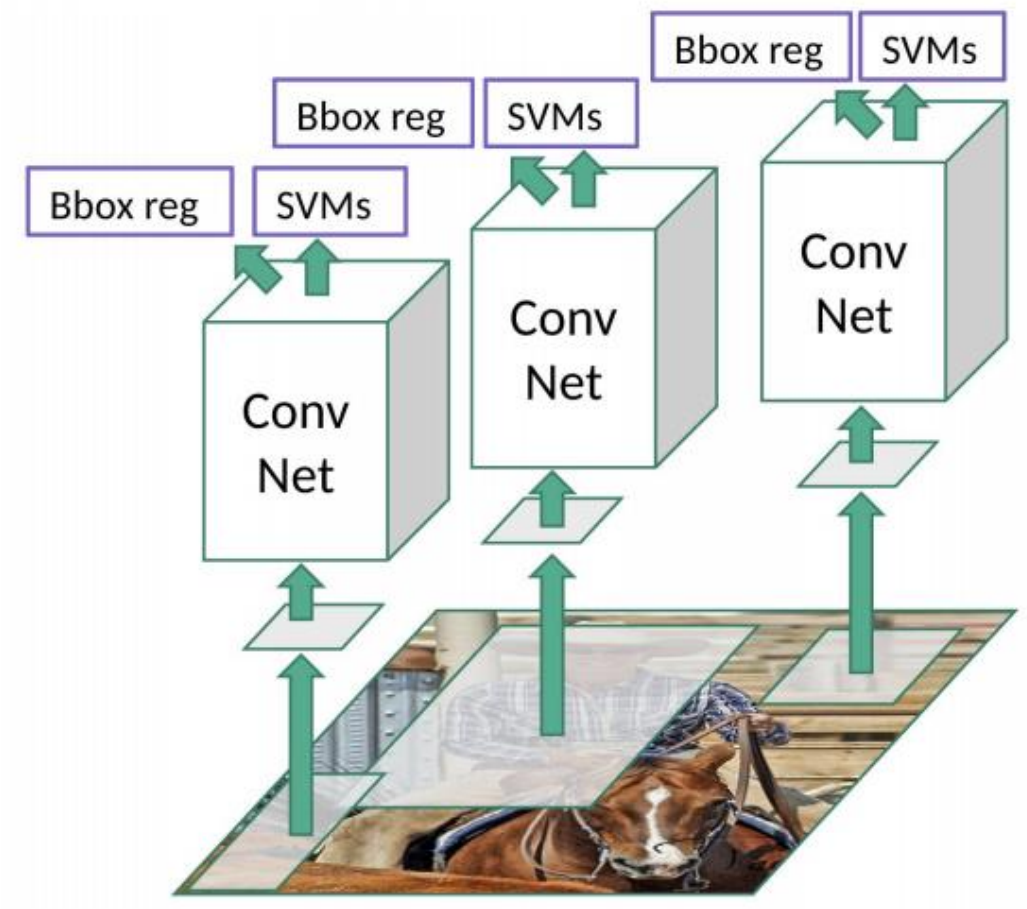
- Non-maximum suppression：剔除同一个目标上的重叠建议框，最终一个目标保留一个得分最高的建议框。

R-CNN简介


R-CNN: *Regions with CNN features*
warped region
CNN
aeroplane? no.
person? yes.
tvmonitor? no.
1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

R-CNN检测流程

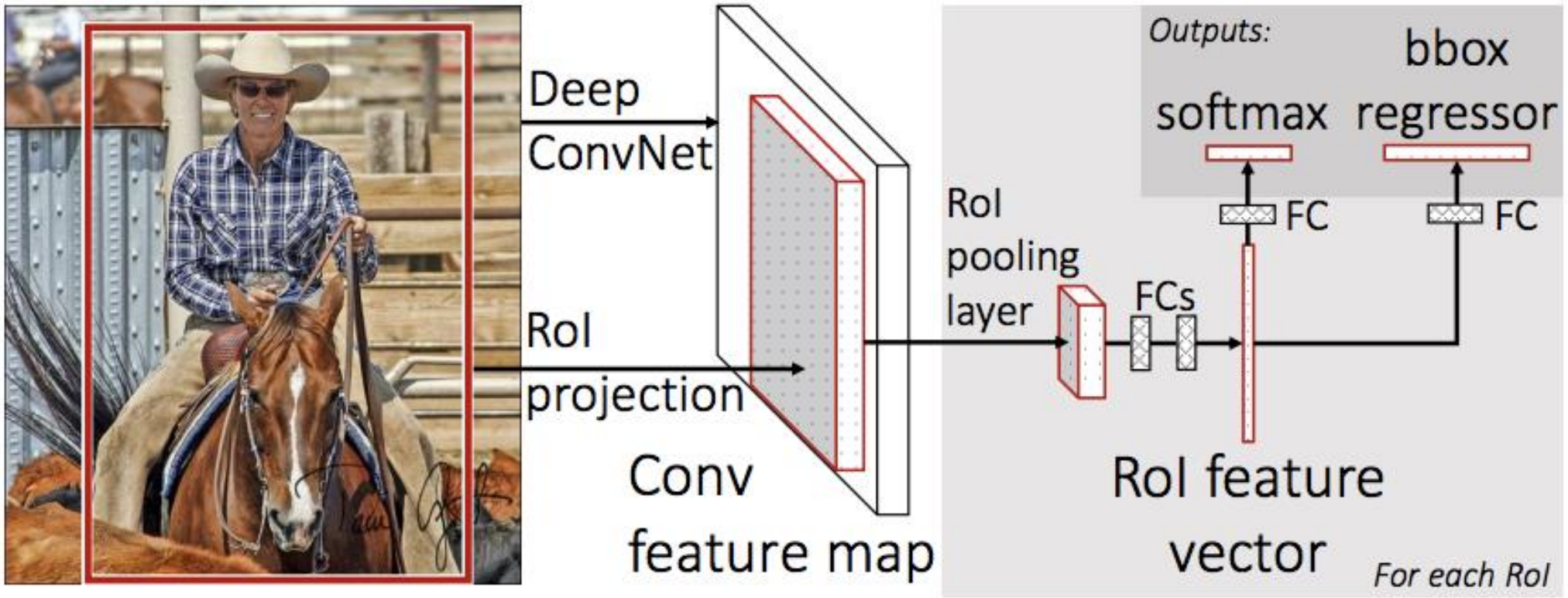
- 将2000个候选区域分别调整成方形
- 由卷积神经网络产生4096维的特征向量；
- 把向量输入到支持向量机进行分类；
- 除此以外，还通过一个Bbox reg预测边界框的4个偏差值，提高边界框的精度。



R-CNN 存在的问题

- 训练过程耗费大量时间（因为每张图像要分类2000个不同区域）
- 不适合实时目标检测（检测每张图像需要约47秒）
- 选择性搜索算法是固定的，不可训练（因此选择性算法可能产生不好的候选区域）

Fast R-CNN

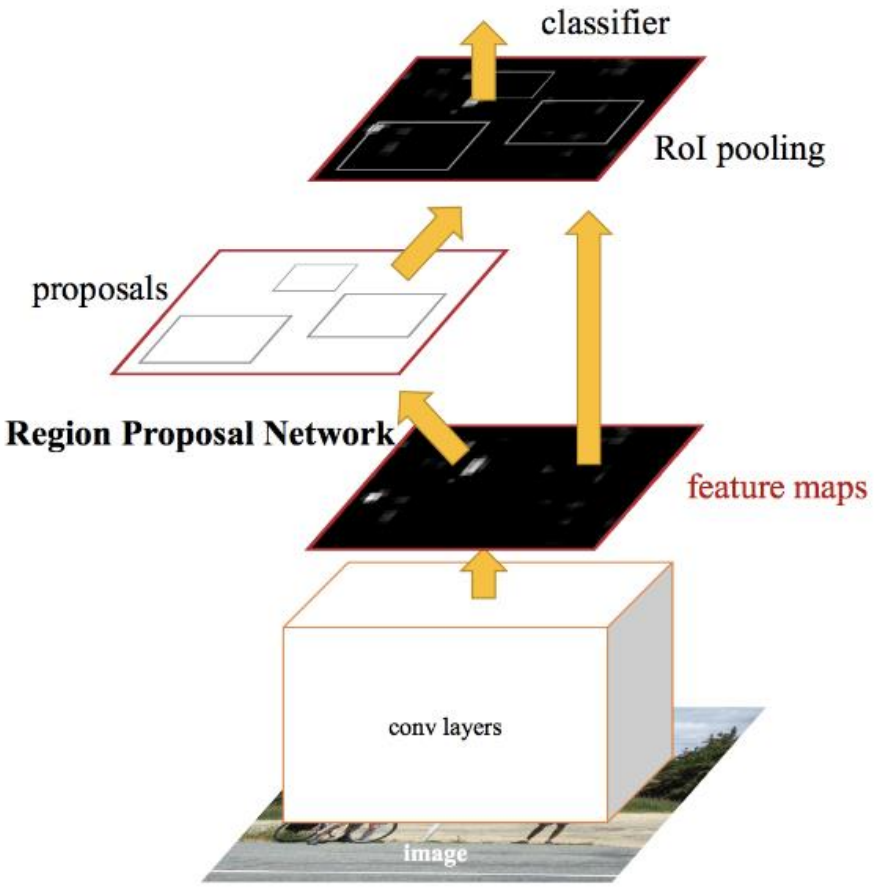

Deep
ConvNet
RoI
projection
Conv
feature map
RoI
pooling
layer
FCs
Outputs:
softmax
bbox
regressor
FC
FC
RoI feature
vector
For each RoI

Fast R-CNN检测过程

- 将原始图像输入卷积神经网络；
- 神经网络产生卷积特征图；
- 从卷积特征图中识别候选区域，并将其变为方形；
- 通过ROI卷积层把区域大小reshape为固定方形大小，喂入全连接层；
- 用softmax进行区域分类，用Bbox回归器预测4个偏差值；

Faster R-CNN

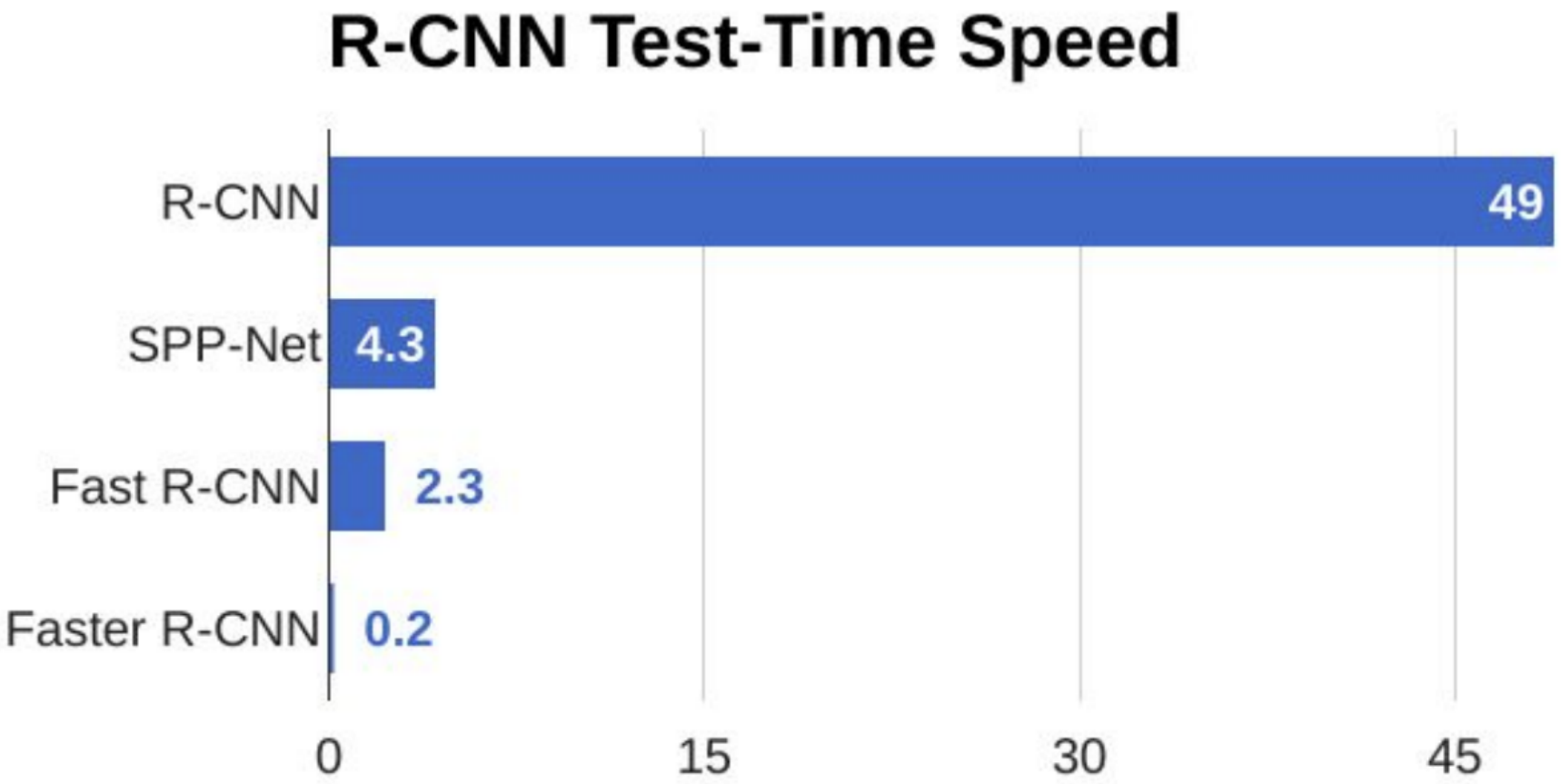
R-CNN和Fast R-CNN都使用了“选择性搜索”算法，这种算法很慢，耗费大量时间。因此Faster R-CNN摒弃了这种算法。



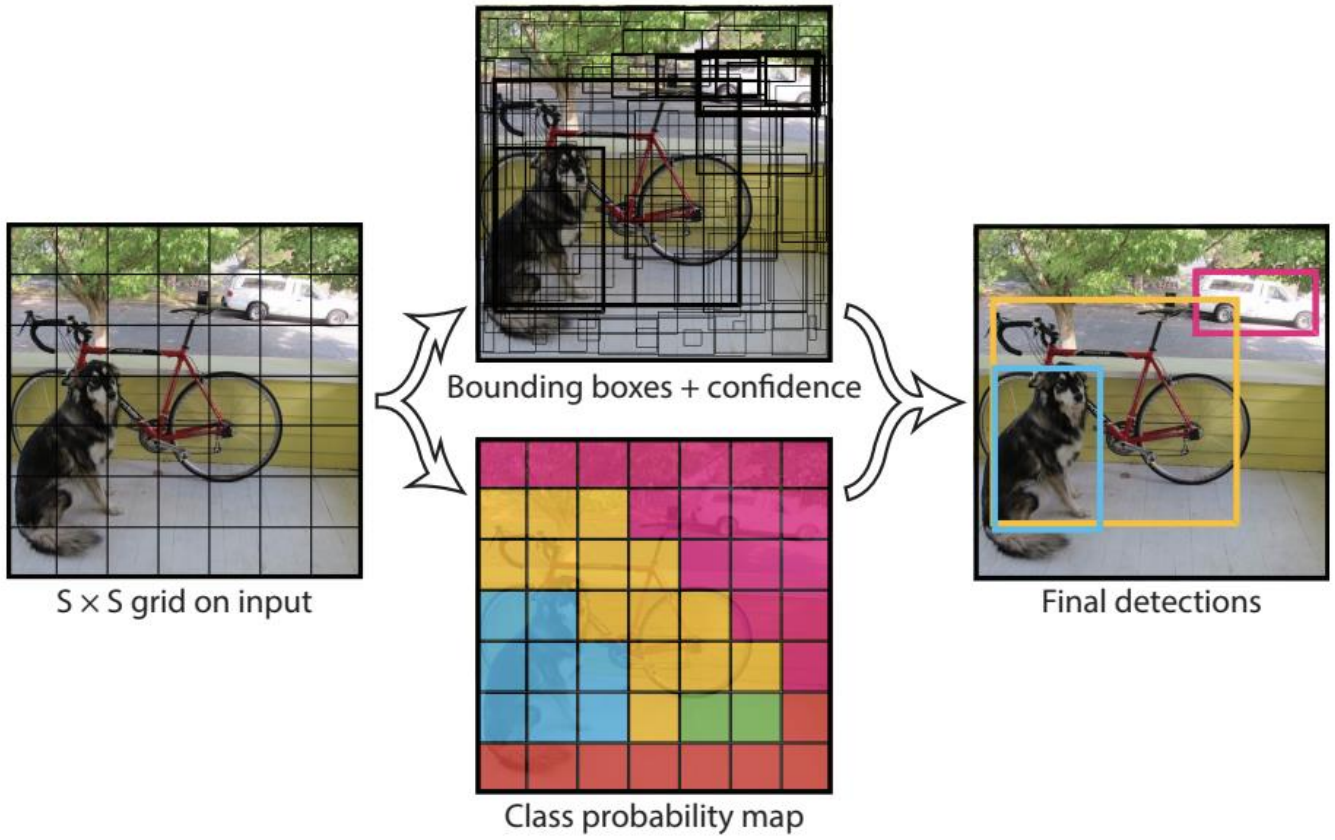
Faster R-CNN检测过程

- 原始图像输入卷积神经网络，生成卷及特征图；
- 通过一个独立的网络预测候选区域；
- 使用ROI池化层把候选区域reshape为固定形状；
- 和Fast R-CNN一样用全连接层分类和预测偏差值。

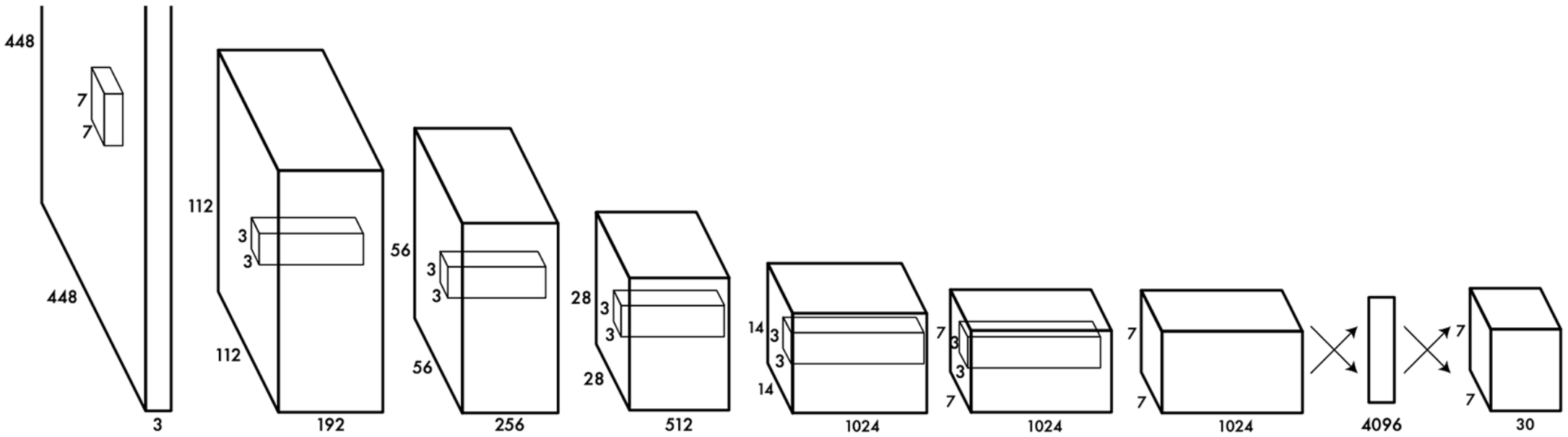
R-CNN系列性能对比


R-CNN Test-Time Speed
R-CNN
SPP-Net
Fast R-CNN
Faster R-CNN
49
4.3
2.3
0.2
0
15
30
45

YOLO：You Only Look Once



YOLO网络结构


448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer
7x7x64-s-2
Maxpool Layer
2x2-s-2
Conv. Layer
3x3x192
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x128
3x3x256
1x1x256
3x3x512
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x256
3x3x512
×4
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x512
3x3x1024
×2
3x3x1024
3x3x1024-s-2
Conv. Layers
3x3x1024
3x3x1024
Conn. Layer
Conn. Layer

YOLO检测方法

- 把原始图像分割为S×S是单元格；
- 每个单元格负责预测m个不同尺度的边界框；
- 每个边界框会预测生成物体种类和偏差值；
- 设置阈值，选取所定位到的物体。

YOLO Loss计算

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{\text{obj}} \cdot \left([(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \right)$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{\text{obj}} \cdot (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{\text{noobj}} \cdot (C_i - \hat{C}_i)^2$$

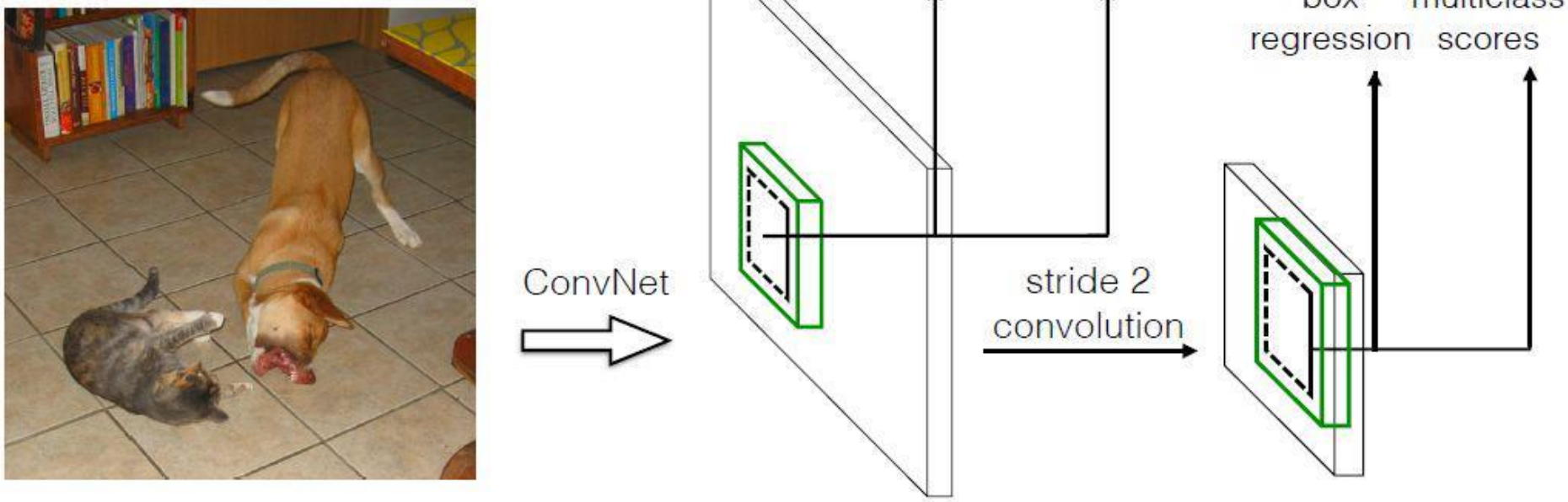
$$+ \sum_{i=0}^{S^2} \mathbb{I}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

YOLO主要问题

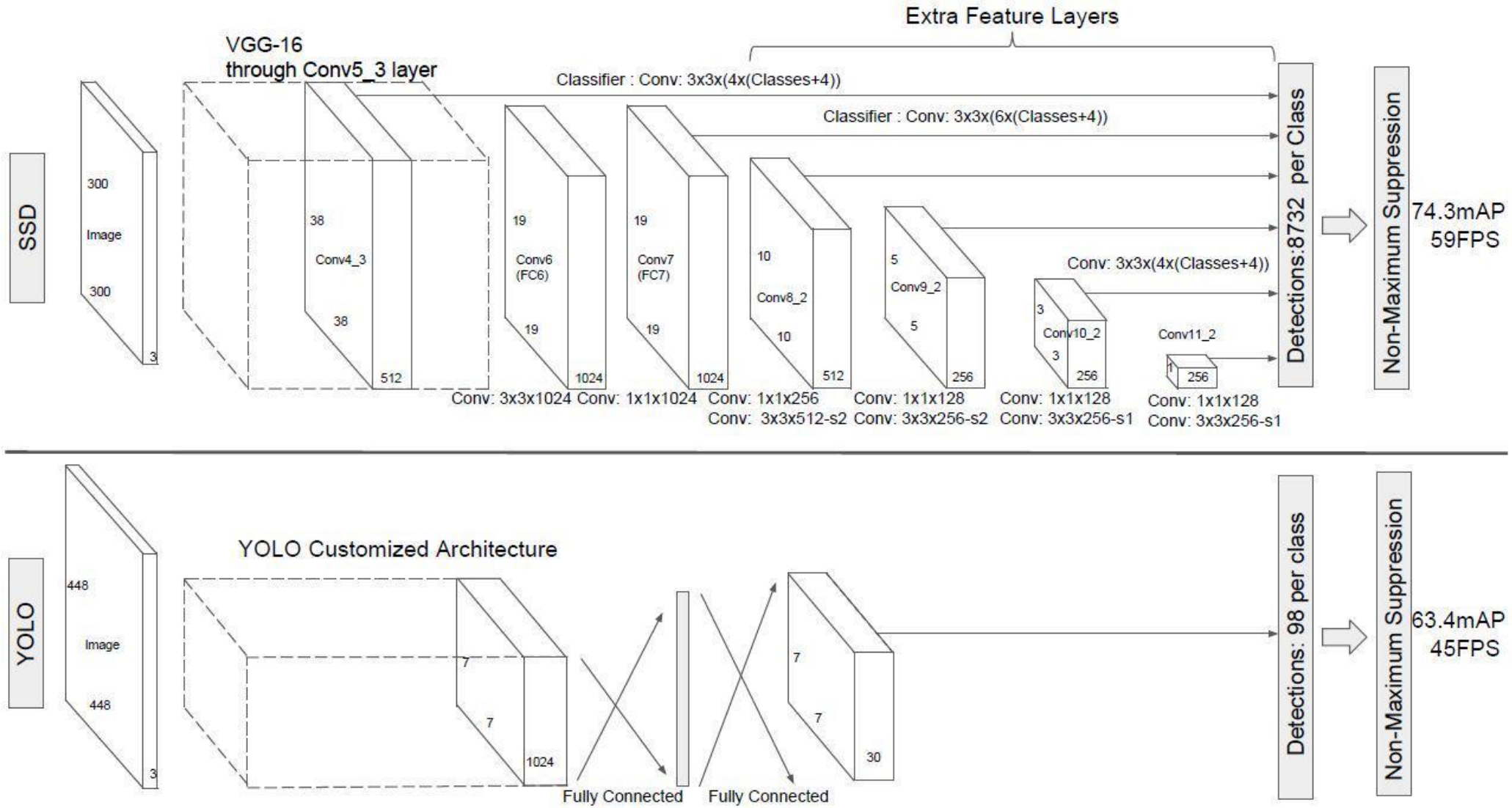
- 对小物体效果不好；
- YOLO 通过下采样后数据得出 BBox 的位置偏移和宽高，所以精度不高，同时对于一些罕见的长宽比或者尺寸的物体的效果不好；
- loss 只是近似反映检测效果的好坏，比如对于不同大小的物体，x, y, w, h 的误差仍不能准确反映 IoU 的误差。YOLO 对于位置的预测较差，是降低 mAP 的主要原因。

SSD ：Signle Shot Detector

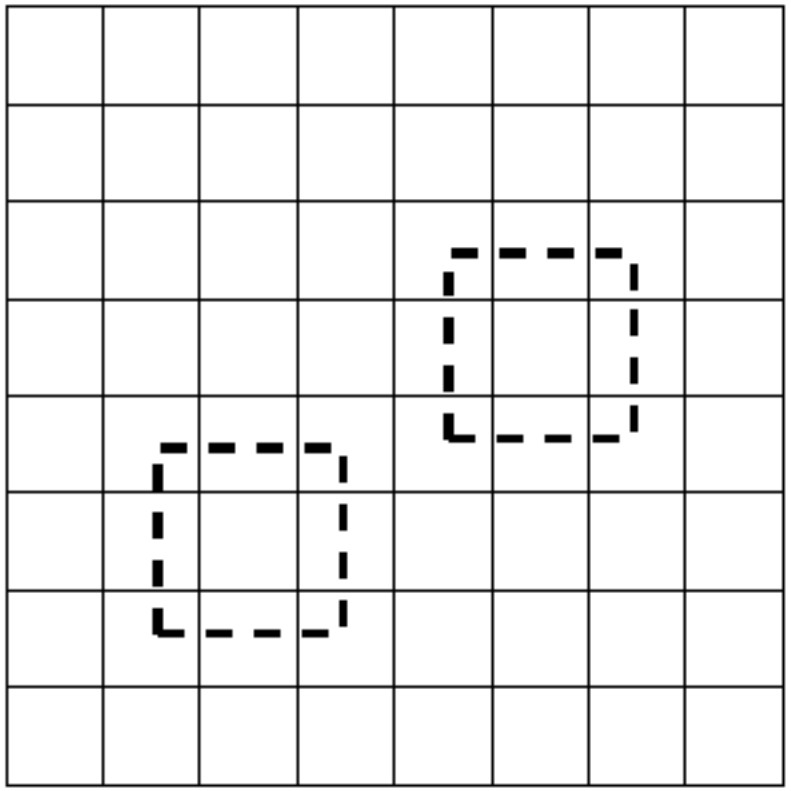
SSD（ 单次检测器）:基于YOLO直接回归边界框和分类概率，借鉴了faster R-CNN的anchor box，使用多尺度特征图检测。用一个神经网络完成目标检测。



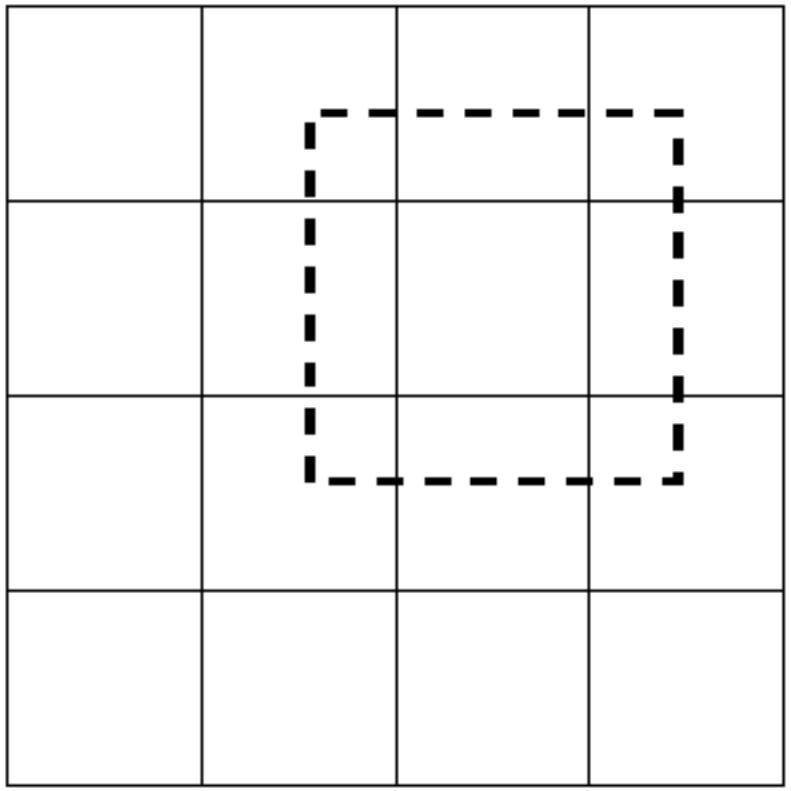
SSD网络结构


Extra Feature Layers
VGG-16
through Conv5_3 layer
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
SSD
300
Image
300
3
38
Conv4_3
38
512
19
Conv6
(FC6)
19
1024
19
Conv7
(FC7)
19
1024
10
Conv8_2
10
512
5
Conv9_2
5
256
3
Conv10_2
3
256
Conv11_2
1
256
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP
59FPS
Conv: 3x3x1024
Conv: 1x1x1024
Conv: 1x1x256
Conv: 3x3x512-s2
Conv: 1x1x128
Conv: 3x3x256-s2
Conv: 1x1x128
Conv: 3x3x256-s1
Conv: 1x1x128
Conv: 3x3x256-s1
YOLO
448
Image
448
3
YOLO Customized Architecture
7
7
1024
Fully Connected
Fully Connected
7
7
30
Detections: 98 per class
Non-Maximum Suppression
63.4mAP
45FPS

SSD多尺度检测



8×8 feature map



4×4 feature map

多尺度：采用大小不同的特征图，CNN网络一般前面的特征图比较大，后面会逐渐采用stride=2的卷积或者pool来降低特征图大小

SSD 网缺陷络

- 需要人工设置prior box的min_size，max_size和aspect_ratio值。网络中prior box的基础大小和形状不能直接通过学习获得，而是需要手工设置。而网络中每一层feature使用的prior box大小和形状恰好都不一样，导致调试过程非常依赖经验。
-
- 虽然采用了pyramdial feature hierarchy的思路，但是对小目标的recall依然一般，并没有达到碾压Faster RCNN的级别。作者认为，这是由于SSD使用conv4_3低级feature去检测小目标，而低级特征卷积层数少，存在特征提取不充分的问题。

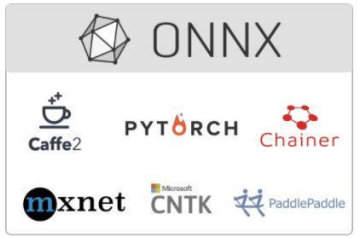
TensorRT简介

- 在推理过程中，基于 TensorRT 的应用程序的执行速度可比 CPU 平台的速度快 40 倍。
- TensorRT 以 NVIDIA 的并行编程模型 CUDA 为基础构建而成。
- TensorRT 针对多种深度学习推理应用的生产部署提供 INT8 和 FP16 优化。

TensorRT框架支持

TensorRT 和 TensorFlow 已紧密集成，因此您可以同时尽享 TensorFlow 的灵活性和 TensorRT 的超强优化性能。有关详情，请参阅 TensorRT 与 TensorFlow 集成 博文。

MATLAB 已通过 GPU 编码器实现与 TensorRT 的集成，这能协助工程师和科学家在使用 MATLAB 时为 Jetson、DRIVE 和 Tesla 平台自动生成高性能推理引擎。有关详情，请参加此在线研讨会。



TensorRT 提供了一个 ONNX 解析器，因此您可以轻松地从框架（例如 Caffe 2、Chainer、Microsoft Cognitive Toolkit、MxNet 和 PyTorch）中将 ONNX 模型导入到 TensorRT。请单击此处，详细了解 TensorRT 中的 ONNX 支持。

TensorRT 还与 ONNX Runtime 集成，助您以 ONNX 格式轻松实现机器学习模型的高性能推理。请单击此处，详细了解 ONNX Runtime 与 TensorRT 的集成。

TensorRT优化核心方法

TensorRT 优化与性能

权重与激活精度校准

通过将模型量化为 INT8 来更大限度地提高吞吐量，同时保持高准确度

层与张量融合

通过融合内核中的节点，优化 GPU 显存和带宽的使用

内核自动调整

基于目标 GPU 平台选择最佳数据层和算法

动态张量显存

更大限度减少显存占用，并高效地为张量重复利用内存

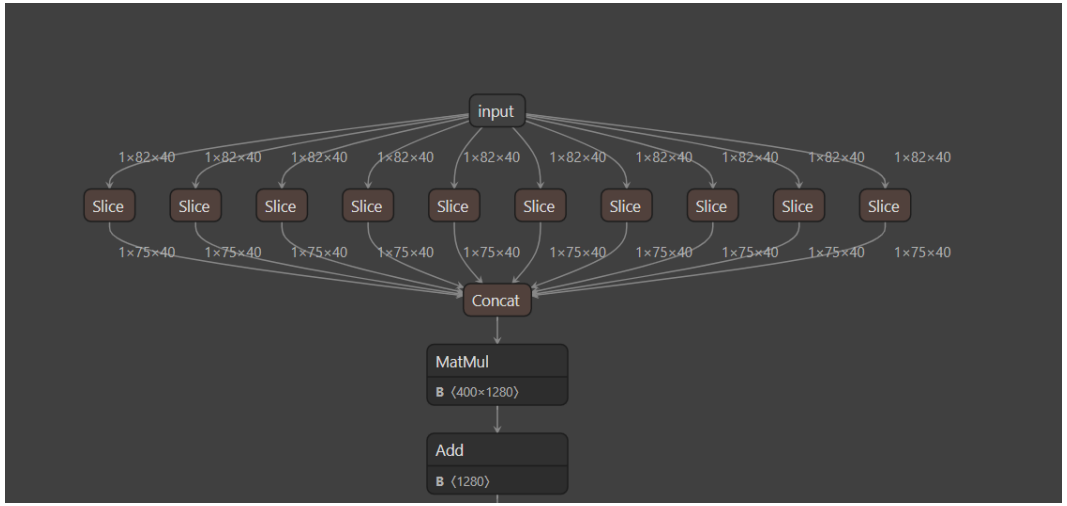
多流执行

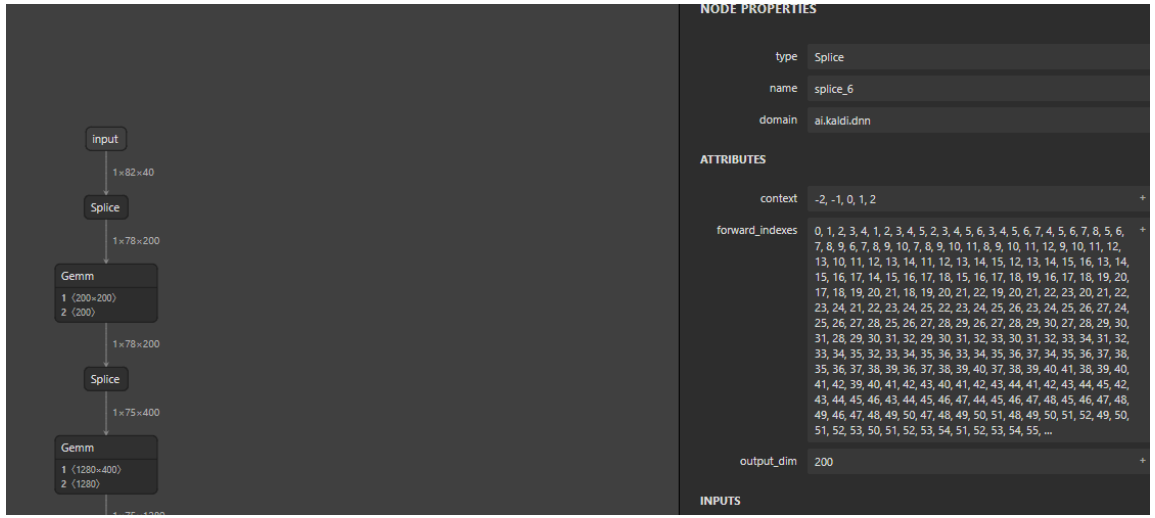
用于并行处理多个输入流的可扩展设计

TensorRT 模型导出

```
import torch
from torch.autograd import Variable
import torch.onnx as torch_onnx
import onnx
def main():
    input_shape = (3, 256, 256)
    model_onnx_path = "unet.onnx"
    dummy_input = Variable(torch.randn(1, *input_shape))
    model = torch.hub.load('mateuszbuda/brain-segmentation-pytorch', 'unet',
      in_channels=3, out_channels=1, init_features=32, pretrained=True)
    model.train(False)

    inputs = ['input.1']
    outputs = ['186']
    dynamic_axes = {'input.1': {0: 'batch'}, '186':{0:'batch'}}
    out = torch.onnx.export(model, dummy_input, model_onnx_path, input_names=inputs, output_names=outputs, dynamic_a
xes=dynamic_axes)

if __name__=='__main__':
    main()
```

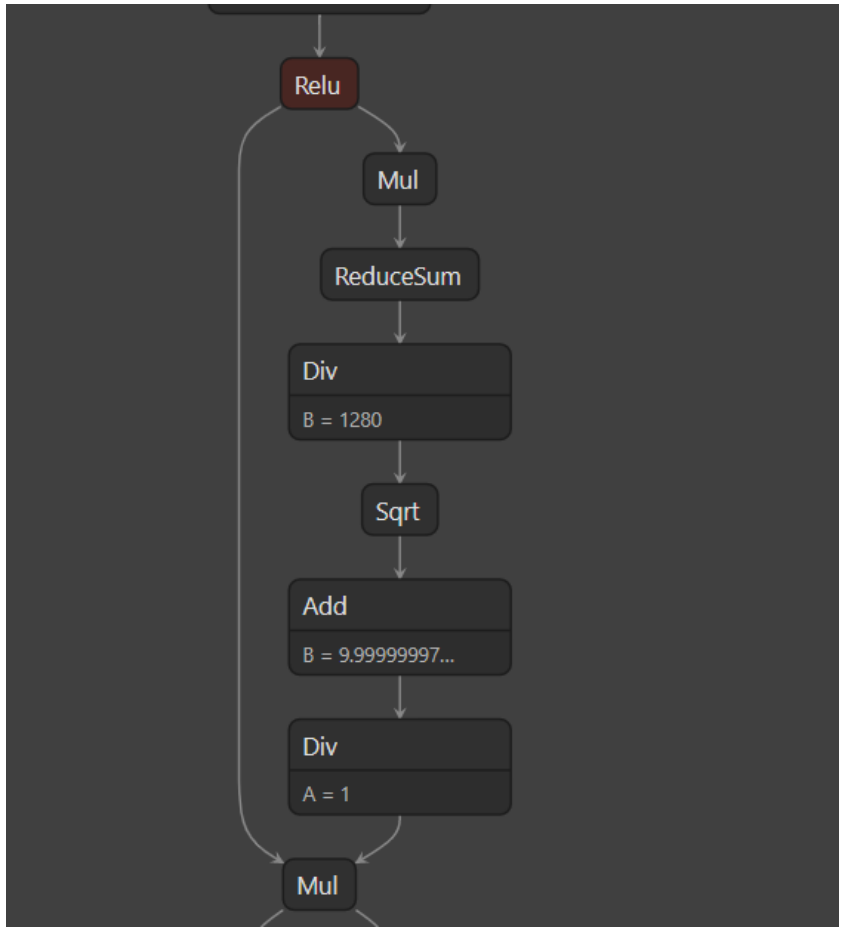
TensorRT Plugin

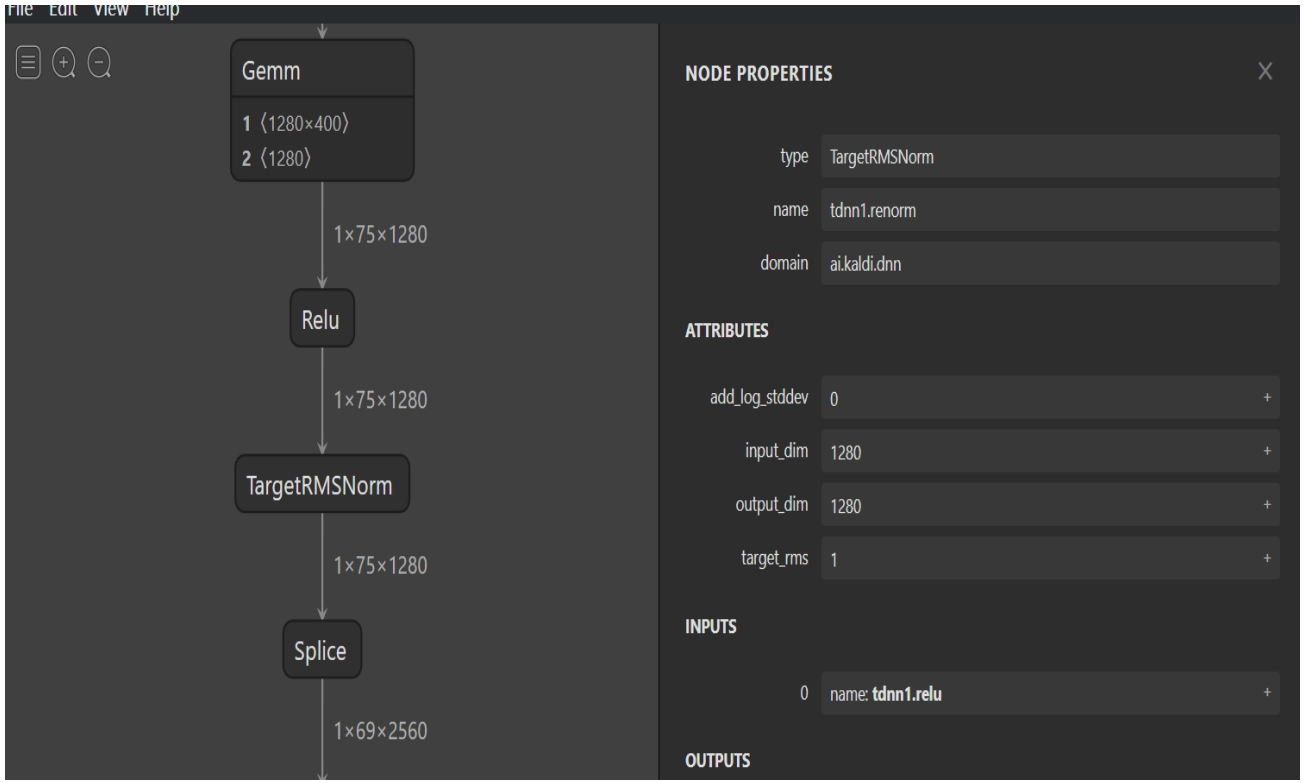




TensorRT Plugin

RMS-Norm的 ONNX 视图：





TensorRT Plugin

开发解析器: parsers/onnx

```
DEFINE_BUILTIN_OP_IMPORTER(TargetRMSNorm)
{
    nvinfer1::ITensor* tensor_ptr = &convertToTensor(inputs.at(0), ctx);
    OnnxAttrs attrs(node);
    int32_t add_log_stddev = attrs.get<int32_t>("add_log_stddev", 0);
    int32_t input_dim = attrs.get<int32_t>("input_dim", 0);
    int32_t output_dim = attrs.get<int32_t>("output_dim", 0);
    float target_rms = attrs.get<float>("target_rms", 1);
    // Populate TargetRMSNorm  plugin properties.
    const std::string pluginName = "TargetRMSNorm_TRT";
    const std::string pluginVersion = "1";
    std::vector<nvinfer1::PluginField> f;
    f.emplace_back("add_log_stddev", &add_log_stddev, nvinfer1::PluginFieldType::kINT32, 1);
    f.emplace_back("input_dim", &input_dim, nvinfer1::PluginFieldType::kINT32, 1);
    f.emplace_back("output_dim", &output_dim, nvinfer1::PluginFieldType::kINT32, 1);
    f.emplace_back("target_rms", &target_rms, nvinfer1::PluginFieldType::kFLOAT32, 1);

    // Create plugin from registry
    nvinfer1::IPluginV2* plugin = importPluginFromRegistry(ctx, pluginName, pluginVersion, node.name(), f);

    ASSERT(plugin != nullptr && "TargetRMSNorm plugin not found in the plugin registry!", ErrorCode::kINTERNAL_ERROR);

    RETURN_FIRST_OUTPUT(ctx->network()->addPluginV2(&tensor_ptr, 1, *plugin));
}
```

TensorRT Plugin

Plugin 创建器：

```
IPluginV2Ext* RmsNormCreator::createPlugin(const char* name, const PluginFieldCollection* fc)
{
    const PluginField* fields = fc->fields;

    RmsNormParam params;

    for (int i = 0; i < fc->nbFields; ++i)
    {
        const char* attrName = fields[i].name;
        int size = fields[i].length;
        if (!strcmp(attrName, "add_log_stddev"))
        {
            params.add_log_stddev = static_cast<int32_t>(*(static_cast<const int32_t*>(fields[i].data)));
        }
        else if (!strcmp(attrName, "input_dim"))
        {
            params.input_dim = static_cast<int32_t>(*(static_cast<const int32_t*>(fields[i].data)));
        }
        else if (!strcmp(attrName, "output_dim"))
        {
            params.input_dim = static_cast<int32_t>(*(static_cast<const int32_t*>(fields[i].data)));
    }
        else if (!strcmp(attrName, "target_rms"))
        {
            params.target_rms = static_cast<float>(*(static_cast<const float*>(fields[i].data)));
        }
    }
    RmsNorm* obj = new RmsNorm(params);
    obj->setPluginNamespace(mNamespace.c_str());
    return obj;
}
```

TensorRT Plugin

TensorRT 算子注册：

```
extern "C" {
bool initLibNvInferPlugins(void* logger, const char* libNamespace)
{
  //User defined tdnn plugin
    initializePlugin<nvinfer1::plugin::BatchNormPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::SplicePluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::RmsNormCreator>(logger, libNamespace);
  initializePlugin<nvinfer1::plugin::GridAnchorPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::NMSPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::ReorgPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::RegionPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::PriorBoxPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::NormalizePluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::RPROIPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::BatchedNMSPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::FlattenConcatPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::CropAndResizePluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::ProposalPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::BatchTilePluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::DetectionLayerPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::ProposalLayerPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::PyramidROIAlignPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::ResizeNearestPluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::SpecialSlicePluginCreator>(logger, libNamespace);
    initializePlugin<nvinfer1::plugin::InstanceNormalizationPluginCreator>(logger, libNamespace);
    return true;
}
} // extern "C"
```

TensorRT Plugin

算子的CUDA Kernel实现：

```cpp
template <typename Real>
__global__ static void TargetRmsNormKernel(
    const Real *x, int x_stride, Real* y, int y_stride, int cols,
  Real target_rms, bool add_log_stddev)
{
    const int i = blockIdx.x;
    const int tid = threadIdx.x;
    const Real* x_row = x + i * x_stride;

    typedef cub::BlockReduce<Real, CU1DBLOCK> BlockReduceT;
    __shared__ typename BlockReduceT::TempStorage temp_storage;

    __shared__ Real stddev_div_target_rms;
    __shared__ Real scale;

    // Reduce x_j^2 to CU1DBLOCK elements per row
    Real tsum = Real(0);
    for (int j = tid; j < cols; j += CU1DBLOCK)
    {
        tsum += x_row[j] * x_row[j];
    }
  //All Reduce to 1
    tsum = BlockReduceT(temp_storage).Sum(tsum);

    if (tid == 0)
    {
        stddev_div_target_rms = sqrt(tsum / (target_rms * target_rms * cols)
    stddev_div_target_rms += 9.999999974752427e-7;
        scale = Real(1) / stddev_div_target_rms;
    }
    __syncthreads();
    // Store normalized input to output
    Real* y_row = y + i * y_stride;
    for (int j = tid; j < cols; j += CU1DBLOCK)
    {
        y_row[j] = x_row[j] * scale;
    }

   if (tid == 0 && add_log_stddev)
   {
       y_row[cols] = log(stddev_div_target_rms * target_rms);
   }
}

```

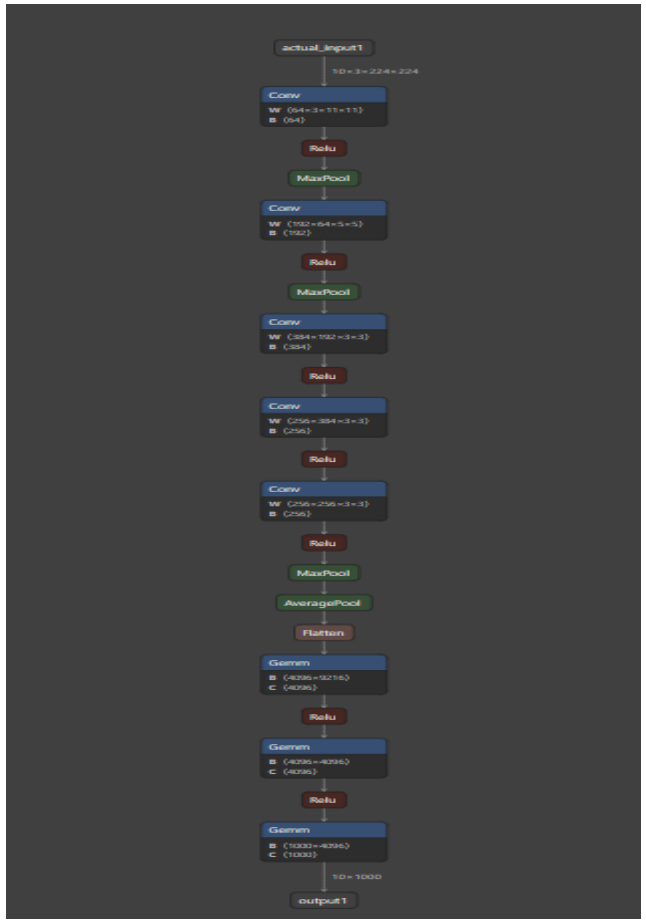
计算图-1 NETRON

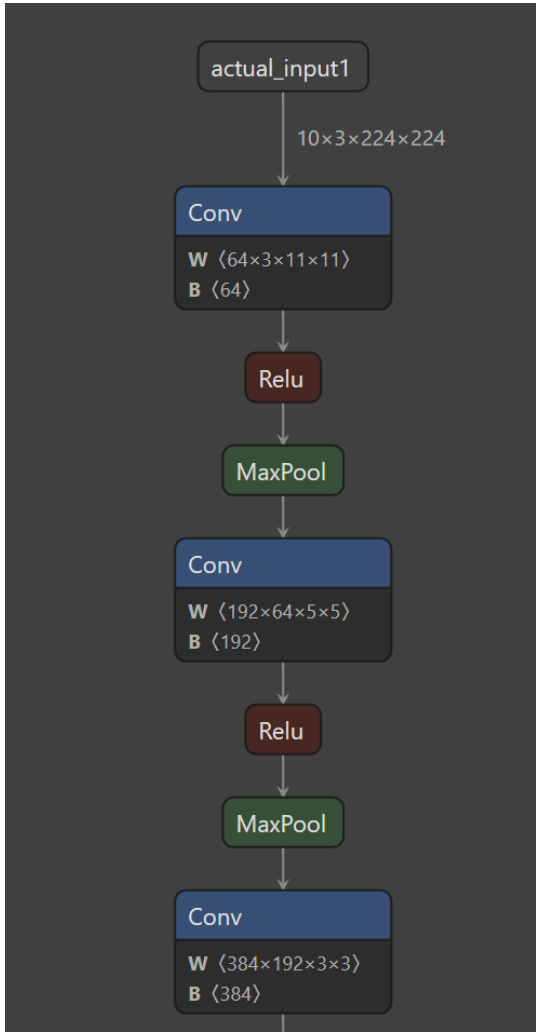
- 项目地址：https://github.com/lutzroeder/Netron

当前支持模型类型：

- ONNX (.onnx, .pb, .pbtxt), Keras (.h5, .keras), Core ML (.mlmodel), Caffe (.caffemodel, .prototxt), Caffe2 (predict_net.pb), Darknet (.cfg), MXNet (.model, -symbol.json), Barracuda (.nn), ncnn (.param), Tengine (.tmfile), TNN (.tnnproto), UFF (.uff) and TensorFlow Lite (.tflite)

计算图-2



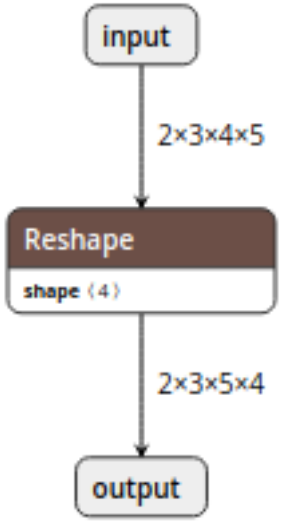

actual_input1
10×3×224×224
Conv
W 〈64×3×11×11〉
B 〈64〉
Relu
MaxPool
Conv
W 〈192×64×5×5〉
B 〈192〉
Relu
MaxPool
Conv
W 〈384×192×3×3〉
B 〈384〉

计算图-3

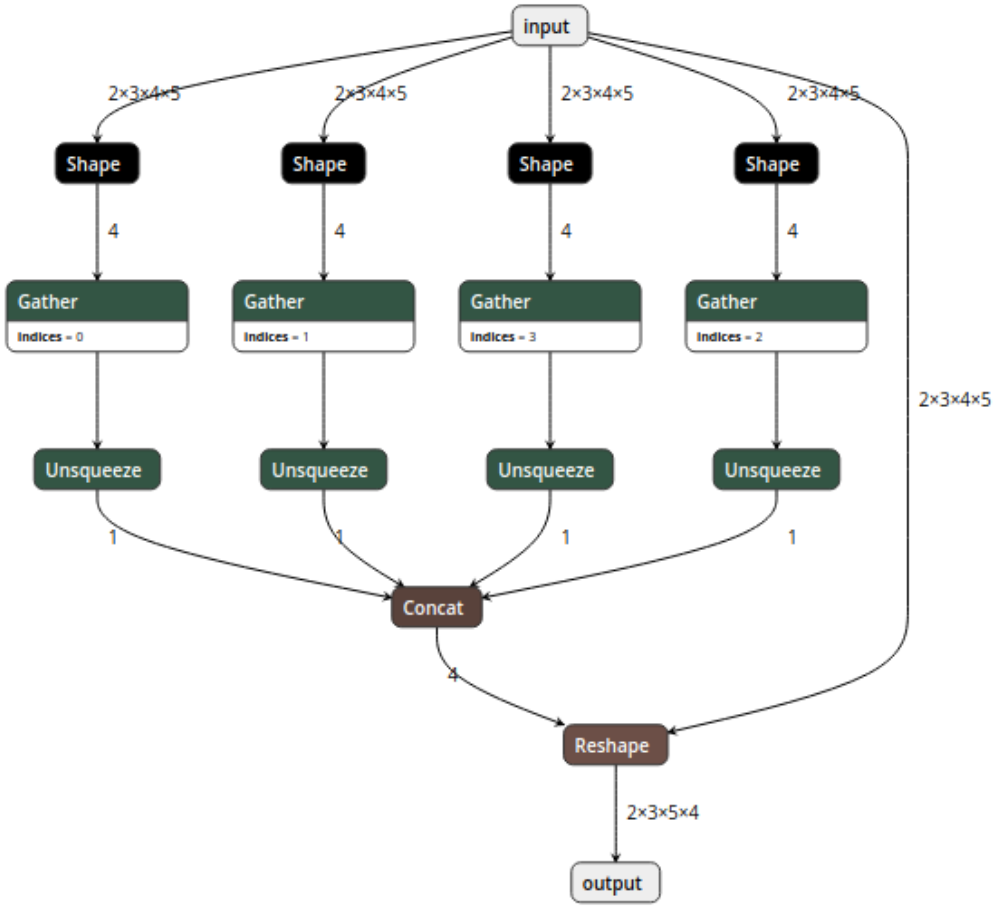
```
import torch


class JustReshape(torch.nn.Module):
    def __init__(self):
        super(JustReshape, self).__init__()

    def forward(self, x):
        return x.view((x.shape[0], x.shape[1], x.shape[3], x.shape[2]))


net = JustReshape()
model_name = 'just_reshape.onnx'
dummy_input = torch.randn(2, 3, 4, 5)
torch.onnx.export(net, dummy_input, model_name, input_names=['input'], output_names=['output'])
```

计算图-4

期待输出：



实际输出：



计算图-5

- 项目地址：https://github.com/daquexian/onnx-simplifier
- 使用方式：

pip3 install onnx-simplifier

- python3 -m onnxsim input_onnx_model output_onnx_model

计算图-6

脚本开发：

```
import onnx
from onnxsim import simplify

# load your predefined ONNX model
model = onnx.load(path + model_name + '.onnx')

# convert model
model_simp, check = simplify(model)

assert check, "Simplified ONNX model could not be validated"

# use model_simp as a standard ONNX model object
```

练习

- 使用TensorRT部署SSD目标检测算法

要求：

（1）Python或C++实现均可。

（2）支持不同层的计算结果输出。

签到&反馈