智能之门

神经网络和深度学习入门

（基于Python的实现）

STEP 6 模型推理部署

第 13 章

模型的推理与部署

模型文件中包含了所有的权重矩阵，记录了该神经网络的数据流图。不同平台、不同语言都可以根据模型文件里的信息构建网络、加载权重矩阵、执行前向计算。

本章先用前面训练好的权重矩阵搭建一个可以手工测试训练效果的应用。然后以问答的形式对模型文件进行介绍，并使用工具观察模型文件的内部结构。接下来以开放式神经网络交换（简称ONNX）格式为例，动手将前面训练好的模型保存为ONNX格式。最后以Windows平台为例，介绍如何在桌面应用中使用ONNX模型。

13.1 手工测试训练效果

神经网络经过训练之后，可以得到一系列能够满足需求的权重矩阵。将输入数据按一定的顺序与权重矩阵进行运算，就可以得到对应的输出。这个过程就是推理的过程。

如果没有保存这些权重矩阵，那么每次使用之前，需要重新训练，这在计算资源有限的情况下是不可取的。另一个选择就是将训练好的权重矩阵保存下来，需要使用的时候重新加载权重矩阵。花费 10 秒钟加载，再花 10 秒钟把结果输出给用户！只需要 20 秒，这个应用就运行结束了。

➢ 训练结果的保存与加载

- 把训练好的网络各层的权重和偏移参数保存到文件中。
- 把这些参数重新加载到网络中，不需要重新训练。

13.1 手工测试训练效果

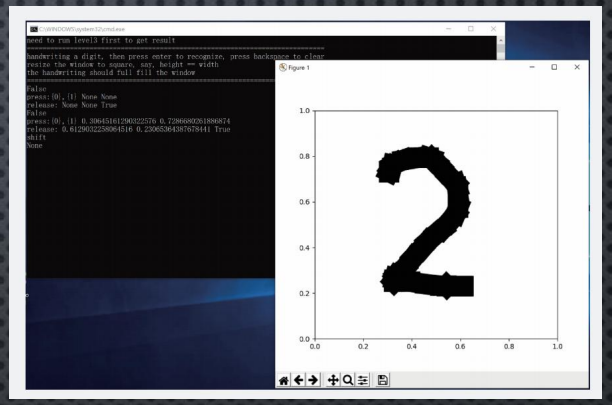
➢ 搭建应用

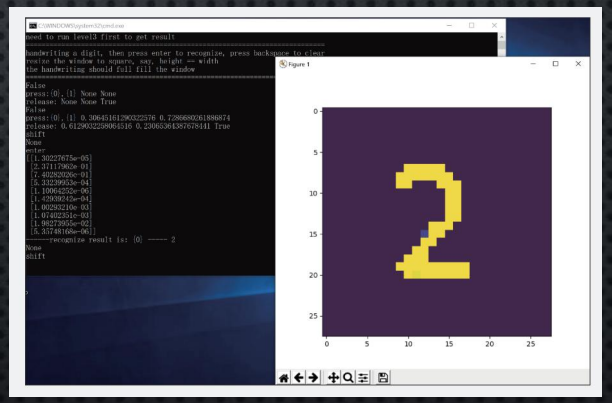
- 搭建交互式应用的实现步骤有：
  - ✓ 重现神经网络结构，加载权重和偏移数据；
  - ✓ 显示界面，让用户可以用鼠标或者手指（触摸屏）写一个数字；
  - ✓ 写好一个数字后，收集数据，转换数据，并触发推理过程；
  - ✓ 得到推理结果；
  - ✓ 清除界面，做下一次测试。

13.1 手工测试训练效果

➢ 交互过程

- 数据处理过程如下：
  - ✓ 应用程序会先把绘图区域保存为一个文件；
  - ✓ 把此文件读入内存，转换成灰度图；
  - ✓ 缩放尺寸到28×28（和训练数据一致）；
  - ✓ 用255减去所有像素值，得到黑底色白前景色的数据（和训练数据一致）；
  - ✓ 归一化到[0,1]（和训练数据一致）；
  - ✓ 变成1×784的数组，调用前向计算方法；
  - ✓ 得到Output后，做一个$argmax$，取到最终结果。





13.2 模型文件概述

加载权重矩阵之前要先搭建网络，而且要和训练时的网络结构一致。所以，网络结构最好也可以像权重矩阵一样保存下来，在需要使用的时候进行加载。

实际上，几乎所有的训练平台也是这么做的，会把网络结构、权重矩阵等信息保存在文件中，这就是我们常说的模型文件，后面也直接简称为模型。

下面我们通过快问快答的形式了解下模型文件。

13.2 模型文件概述

➢ 为什么需要模型文件?

- 如今人工智能发展的越来越快，在图像分析、自然语言处理、语音识别等领域都有着令人惊喜的效果。而模型，可以想象为一个“黑盒”，输入是你需要处理的一张图像，输出是一个它的类别信息或是一些特征，模型文件也因此保存了能完成这一过程的所有重要信息，并且还能用来再次训练、推理等，方便了模型的传播与发展。

➢ 模型文件描述的是什么?

- 首先我们需要了解，目前绝大部分的深度学习框架都将整个AI模型的计算过程抽象成数据流图，用户写的模型构建代码都由框架组建出一个数据流图（也可以简单理解为神经网络的结构），而当程序开始运行时，框架的执行器会根据调度策略依次执行数据流图，完成整个计算。当有了这个背景知识后，我们很容易想到，为了方便地重用AI模型的计算过程，我们需要将它运行的数据流图、相应的运行参数和训练出来的权重保存下来，这就是AI模型文件主要描述的内容。

13.2 模型文件概述

- **AI模型的作用是什么?**
  - 以视觉处理为例，人通过眼睛捕获光线，传递给大脑处理，返回图像的一些信息，比如，这是花，是动物。AI模型的作用就相当于大脑的处理，能根据输入的数据给予一定的判断。使用封装好的AI模型，那么设计者只需要考虑把输入的数据处理成合适的格式（类似于感光细胞的作用），然后传递给AI模型（大脑），之后就可以得到一个想要的输出。
- **模型文件有哪些类型，TensorFlow和其他框架的有什么区别?**
  - 由于每个深度学习框架都有自己的设计理念和工具链，对数据流图的定义和粒度都不一样，所以每家的AI模型文件都有些区别，几乎不能通用。例如，TensorFlow的Checkpoint Files用Protobuf去保存数据流图，用SSTable去保存权重；Keras用Json表述数据流图而用h5py去保存权重；PyTorch由于是主要聚焦于动态图计算，模型文件甚至只用pickle保存了权重而没有完整的数据流图。TensorFlow在设计之初，就考虑了从训练、预测、部署等复杂的需求，所以它的数据流图几乎涵盖了整个过程可能涉及到操作，例如初始化、后向求导及优化算法、设备部署和分布式化、量化压缩等，所以只需要通过TensorFlow的模型文件就能够获取模型完整的运行逻辑，所以很容易迁移到各种平台使用。

13.2 模型文件概述

- **我拿到了别人的一个模型文件，我自己有一些新的数据，就能继续训练AI么？如果不能，还差什么呢?**
  - 训练模型的时候，除了网络架构和权重，还有训练时所使用的各种超参，比如使用的优化器（Optimizer）、批量大小（Batch size）、学习率（Learning rate）、冲量（Momentum）等，这些都会影响我们再训练的效果，需要格外注意。
- **ONNX文件是什么，如何保存为ONNX文件?**
  - 开放式神经网络交换是由微软、FaceBook、亚马逊等多个公司一起推出的，针对机器学习设计的开放式文件格式，可以用来存储训练好的模型。它使得不同的人工智能框架可以采用相同格式存储模型数据并交互。目前很多机器学习框架都支持ONNX格式，如PyTorch、Caffe2、CNTK、ML.NET、MXNet等，它们都有专门的export_to_onnx方法，通过遍历它们原生的数据流图，转化为ONNX标准的数据流图。而对于TensorFlow这样并不原生支持ONNX的框架，通常会使用图匹配的方式转化数据流图。

13.2 模型文件概述

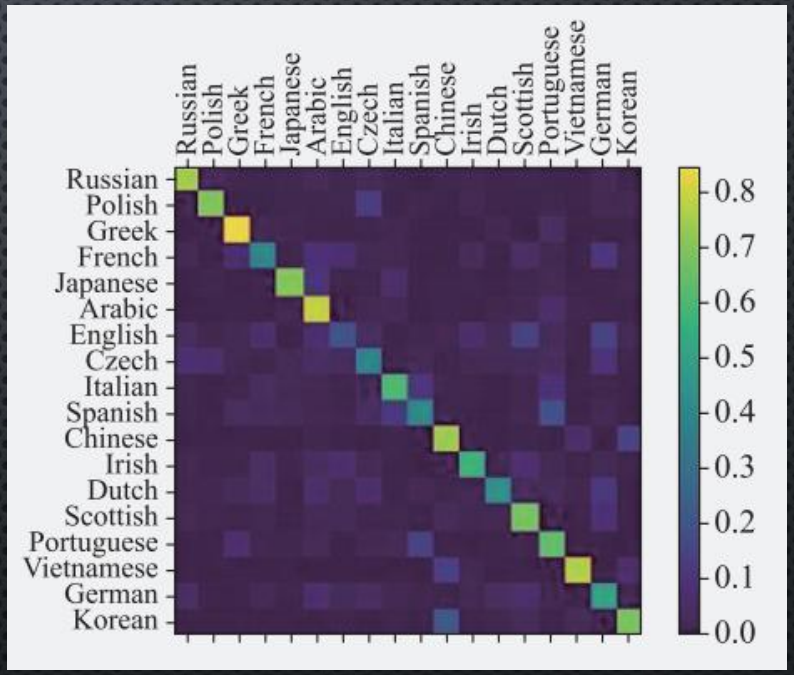
- **转化得来的模型文件有什么信息丢失么?**
  - 由于模型文件仅仅描述了数据流图和权重，并不包含操作符（Op）的具体实现，所以不同框架对于“同名”的操作符理解和实现也会有所不同（例如不同框架的Conv/Pool Padding方式），最终有可能得到不完全一致的推理结果。
- **模型文件是如何与应用程序一起工作的?**
  - 应用程序使用模型文件，本质也是要执行模型文件的数据流图。一般有两种方式实现模型文件和应用程序的协作：如果有可以独立执行模型文件的运行时，例如系统级别的CoreML、WinML和软件级别的Caffe、DarkNet等，我们就可以在程序中动态链接直接使用；除此以外，我们也可以将数据流图和执行数据流图的程序（一般称为Op Kernel）编译在一起，从而脱离运行时，由于单一模型涉及到的操作有限，这样可以极大减少框架所占用的资源。在将模型集成到应用程序中前，应该先使用模型查看工具（如Netron等）查看模型的接口、输入输出的格式和对应的范围，然后对程序中传入模型的输入作对应的预处理工作，否则可能无法得到预期的效果。

13.2 模型文件概述

- **如果我本地机器有GPU，那么我在运行推理模型的时候，怎么能利用上本地的资源呢?**
  - 首先需要安装匹配的显卡驱动、CUDA和GPU版的框架，然后根据框架进行代码调整：对于TensorFlow这样能够自动做设备部署的框架，它会尽量把GPU支持的操作自动分配给GPU计算，不太需要额外的适配；对于PyTorch、MXNet这样不具有自动设备部署功能的框架，可能需要进行一个额外的操作将模型、张量从CPU部署到GPU上。
- **如果一个模型文件已经集成到一个应用程序中发布到了很多用户那里， 这时候我们又训练出了一个新的模型，怎么更新众多的应用达到持续开发和持续集成的效果呢?**
  - 如果应用程序是依赖额外的运行来使用模型，只需要更新模型文件就可以了；如果是使用的模型和Kernel编译在一起的方式，就需要重新编译程序。

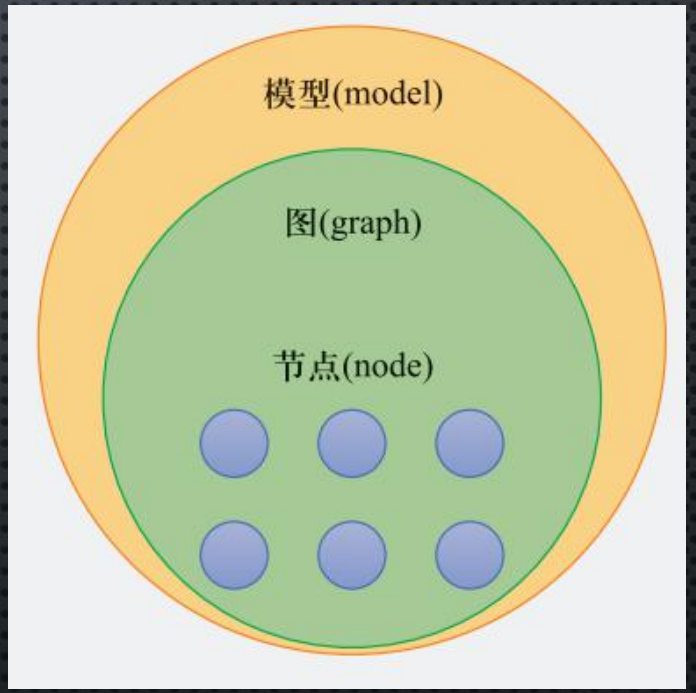
13.2 模型文件概述

- **模型文件有单元测试来保证质量标准么？**
  - 在机器学习领域，混淆矩阵可以用来呈现算法性能的可视化效果，通常被用来显示监督学习的效果。混淆矩阵的名字来源于它可以非常容易的表明多个类别是否有混淆，也就是一种类别是否被预测成另一类别。右图是混淆矩阵可视化的效果。
  - 程序员们训练了一轮模型，经过测试，该模型在大部分测试样例上表现的很好，但有个别的表现不好。于是经过了对不好的样例的分析，对模型进行调整和重新训练。也许重新训练后在这些特定的例子上准确率已经很高，但是我们无法确认，新的模型是否在原来已经预测很准确的例子上仍然表现良好，这时就引入了混淆矩阵。



13.3 ONNX模型文件

ONNX 是一个开放式的规范，定义了可扩展的计算图模型、标准数据类型以及内置的运算符。该文件在存储结构上可以理解为是一种层级的结构，右图描述了 ONNX模型文件的简化结构。最顶层结构是模型（model），模型记录了该模型文件的基本属性，如使用的 ONNX标准的版本、使用的运算符集版本、制造商的名字和版本等信息，除此以外，模型中记录着最主要的信息是图（graph）。



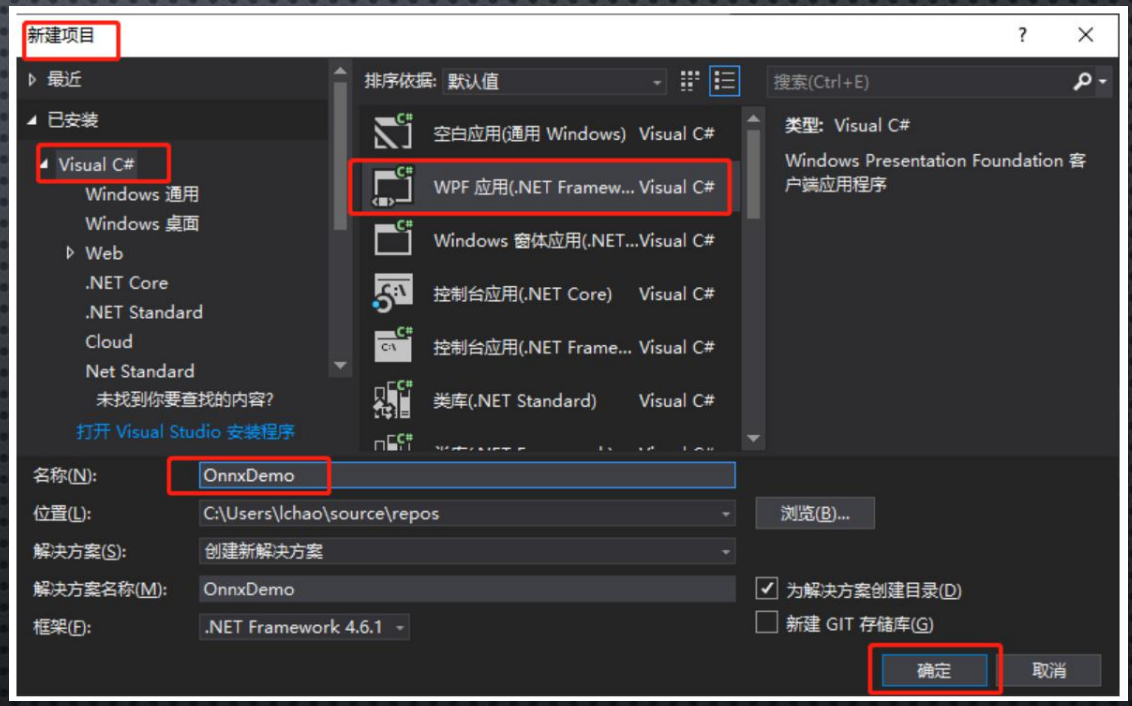
图可以理解为是计算图的一种描述，是由输入、输出以及节点组成的（右图省略了输入与输出），它们之间通过寻找相同的名字实现连接，也就是说，相同名字的变量会被认为是同一个变量，如果一个节点的输出名字和另一个节点的输入名字相同，这两个节点会被认为是连接在一起的。节点（node）就是要调用的运算符，多个节点以列表的形式在图中存储。ONNX 支持的运算符类型可以在 ONNX 官方文档中查看。

13.3 ONNX模型文件

- **创建ONNX节点**
  - ONNX采用了Protobuf（Google Protocol Buffer）格式进行存储，这种格式是是一种轻便高效的结构化数据存储格式，可以用于结构化数据序列化，很适合做数据存储或数据交换格式。Protobuf在使用时需要先在proto文件中定义数据格式，然后构造相关的对象。Python中的onnx库已经提供了创建ONNX的帮助类，使用起来很方便。
- **创建ONNX文件**
  - 有了节点列表后，就可以创建对应的图（Graph），最终创建出模型（Model）。
- **保存多入多出三层神经网络**
  - 前面章节中，我们训练了多入多出的三层神经网络来完成MNIST数据集的识别数字任务，现在就可以动手把训练好的网络保存为ONNX格式的模型文件。

13.4 Windows中模型的部署

微软开源了ONNX运行时库（ONNX Runtime），可以进行高性能的推理，支持主流三大操作系统平台，支持使用GPU，同时提供Python、C#、C/C++、Ruby等开发语言，能满足各种场景的使用。



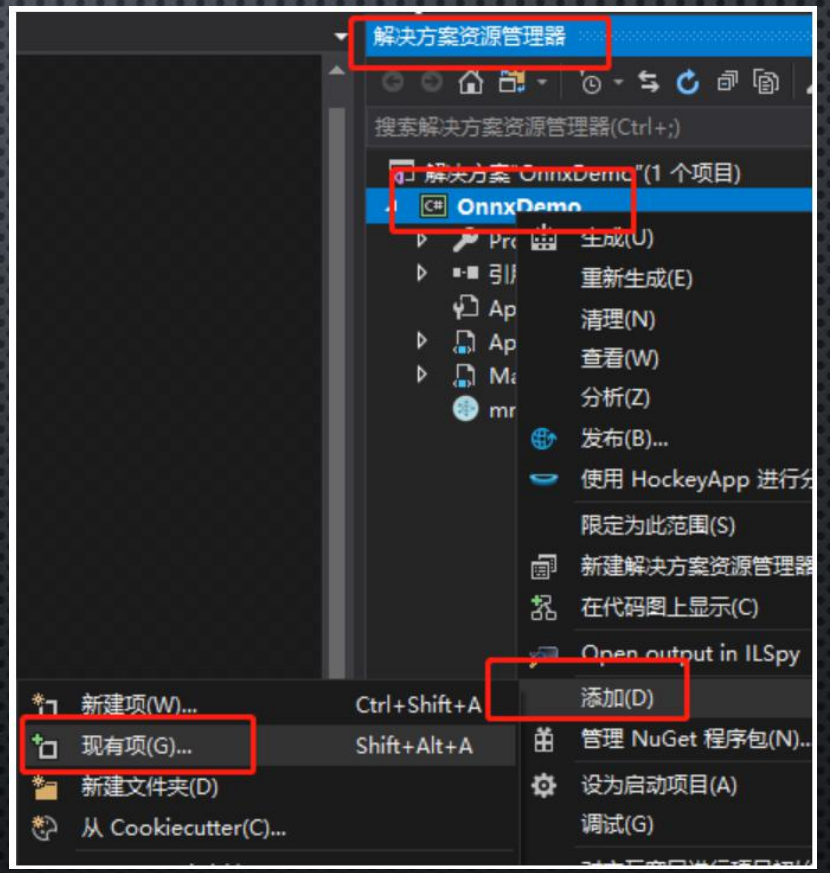
➢ 创建WPF项目

- 打开Visual Studio 2017，新建项目，在Visual C#分类中选择“WPF应用”，填写项目名称OnnxDemo，点击确定，即可完成一个空白项目的创建。

13.4 Windows中模型的部署

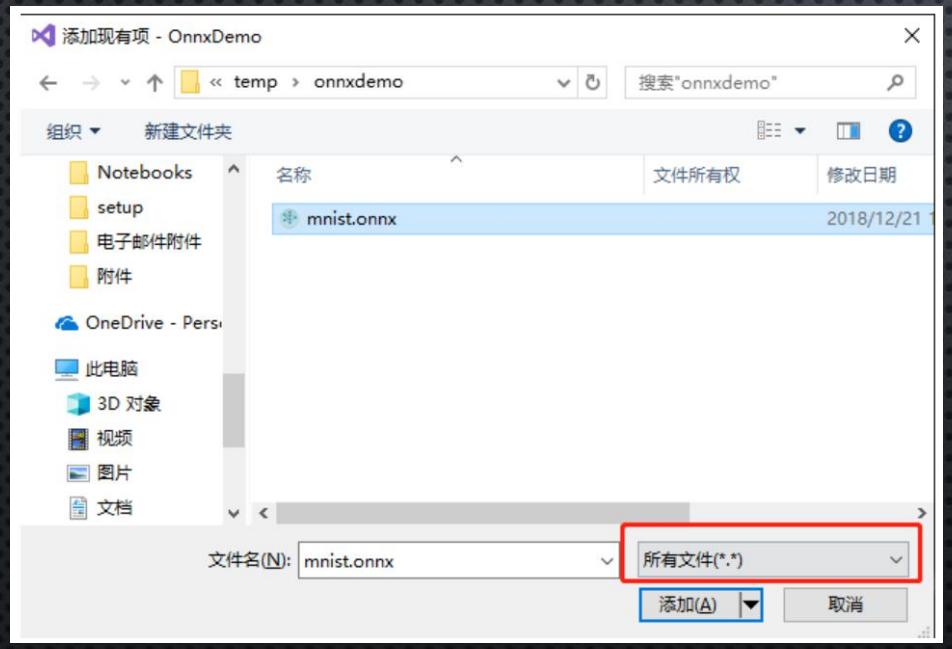
➢ 添加模型文件到项目中

- 打开解决方案资源管理器中，在项目上点右键->添加->现有项。



13.4 Windows中模型的部署

- 在弹出的对话框中，将文件类型过滤器改为所有文件，然后导航到模型所在目录，选择模型文件并添加。

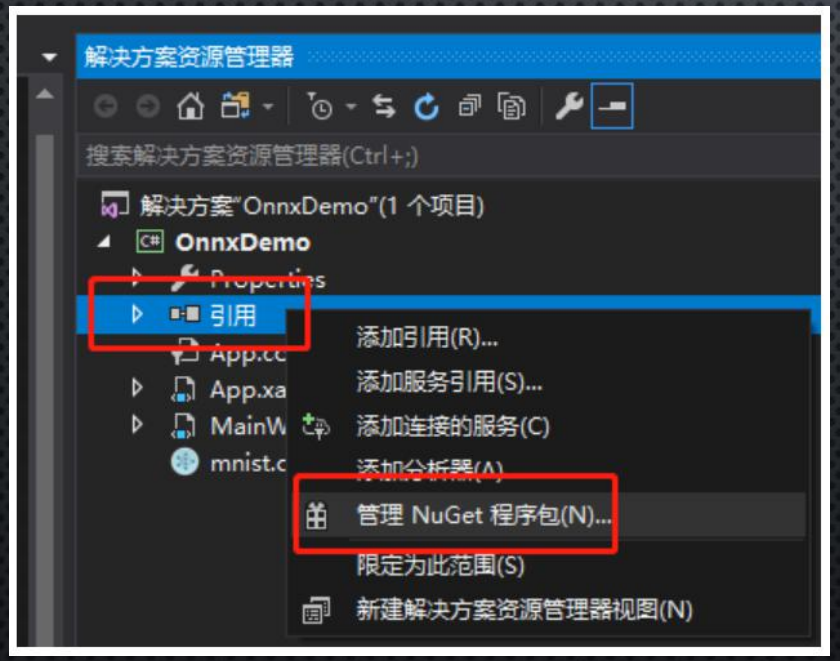


- 模型是在应用运行期间加载的，所以在编译时需要将模型复制到运行目录下。在模型文件上点右键->属性，然后在属性面板上，将“生成操作”属性改为“内容”，将“复制到输出目录”属性改为“如果较新则复制”。

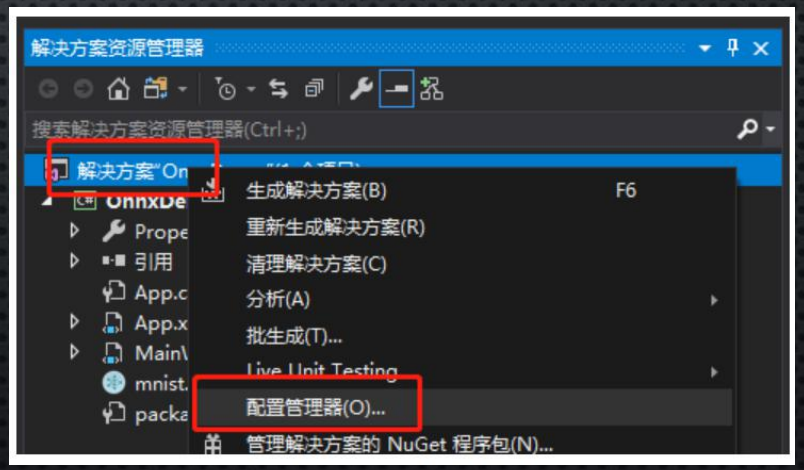
13.4 Windows中模型的部署

➢ 添加OnnxRuntime库

- 打开解决方案资源管理器，在引用上点右键，管理NuGet程序包。

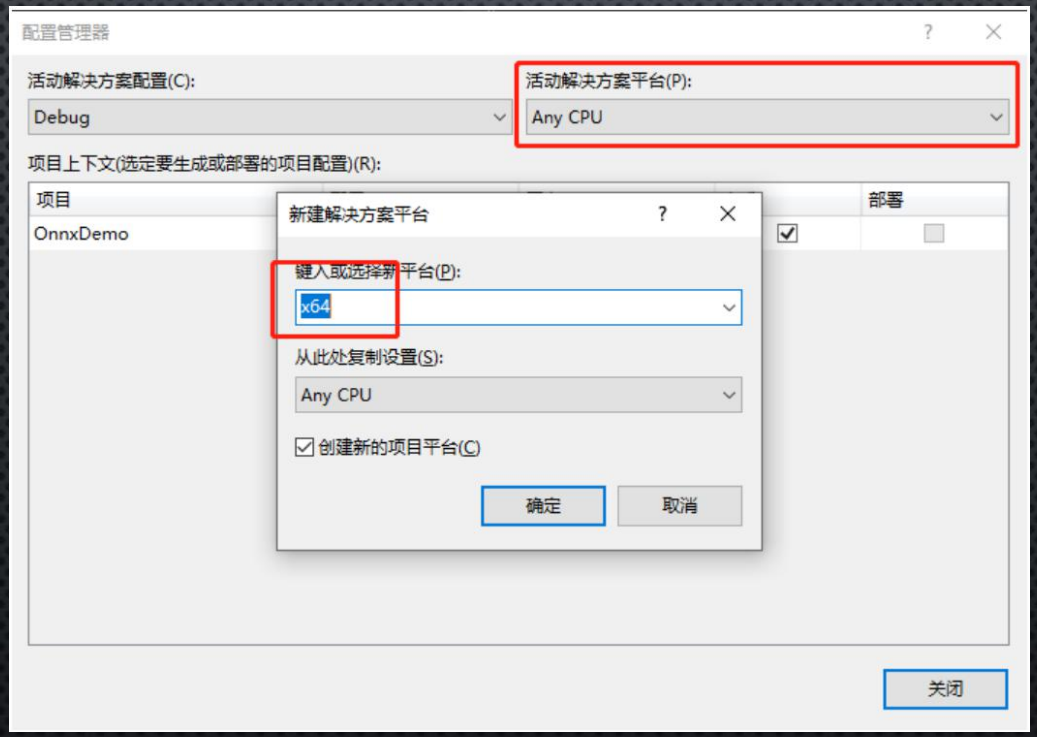


- 在打开的NuGet包管理器中，切换到浏览选项卡，搜索onnxruntime，找到Microsoft.ML.OnnxRuntime包，当前版本是1.0.0，点击安装，按提示完成安装即可。该版本不支持AnyCPU平台，所以需要将项目的目标架构显式的改为x64或x86。在解决方案上点右键，选择配置管理器。



13.4 Windows中模型的部署

- 在配置管理器对话框中，将活动解决方案平台切换为x64或x86。如果没有x64和x86，在下拉框中选择新建，按提示新建x64或x86平台。

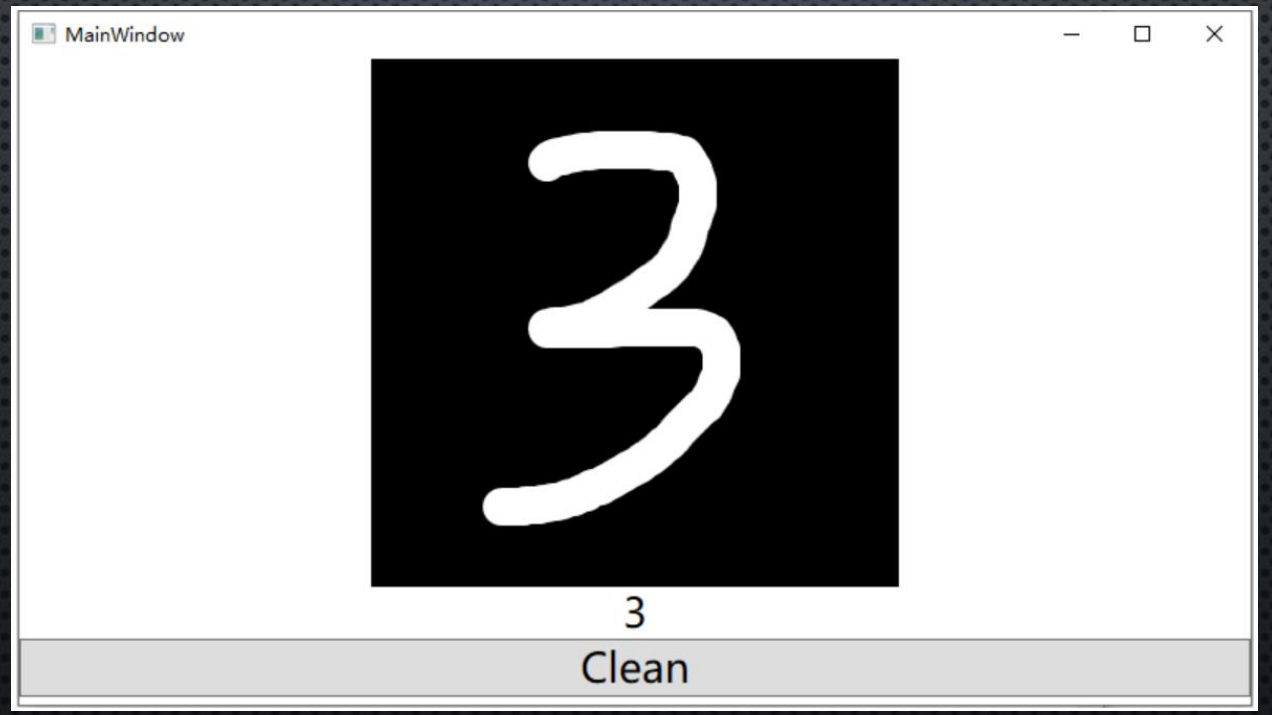


13.4 Windows中模型的部署

- **设计界面**
  - 打开MainWindow.xaml，替换整个Grid片段的代码；然后在MainWindow构造函数中调用InitInk方法初始化画布，设置画笔颜色为白色；然后添加btnClean按钮事件的实现。
- **画布数据预处理**
  - 这里需要添加几个函数对画布数据进行处理，转为模型可以接受的数据。以下几个函数分别是将画布渲染到28×28的图片，读取每个像素点的值，生成模型需要数组。
- **调用模型进行推理**
  - 整理好输入数据后，就可以调用模型进行推理并输出结果了。

13.4 Windows中模型的部署

- **结果展示**
  - 运行程序并书写数字，即可看到模型推理结果。



THE END

谢谢！