20191022 ---- 第六次作业

一、基于MNIST数据集的手写数字识别应用开发实践

1、构建项目

①、程序分析：

步骤如下：

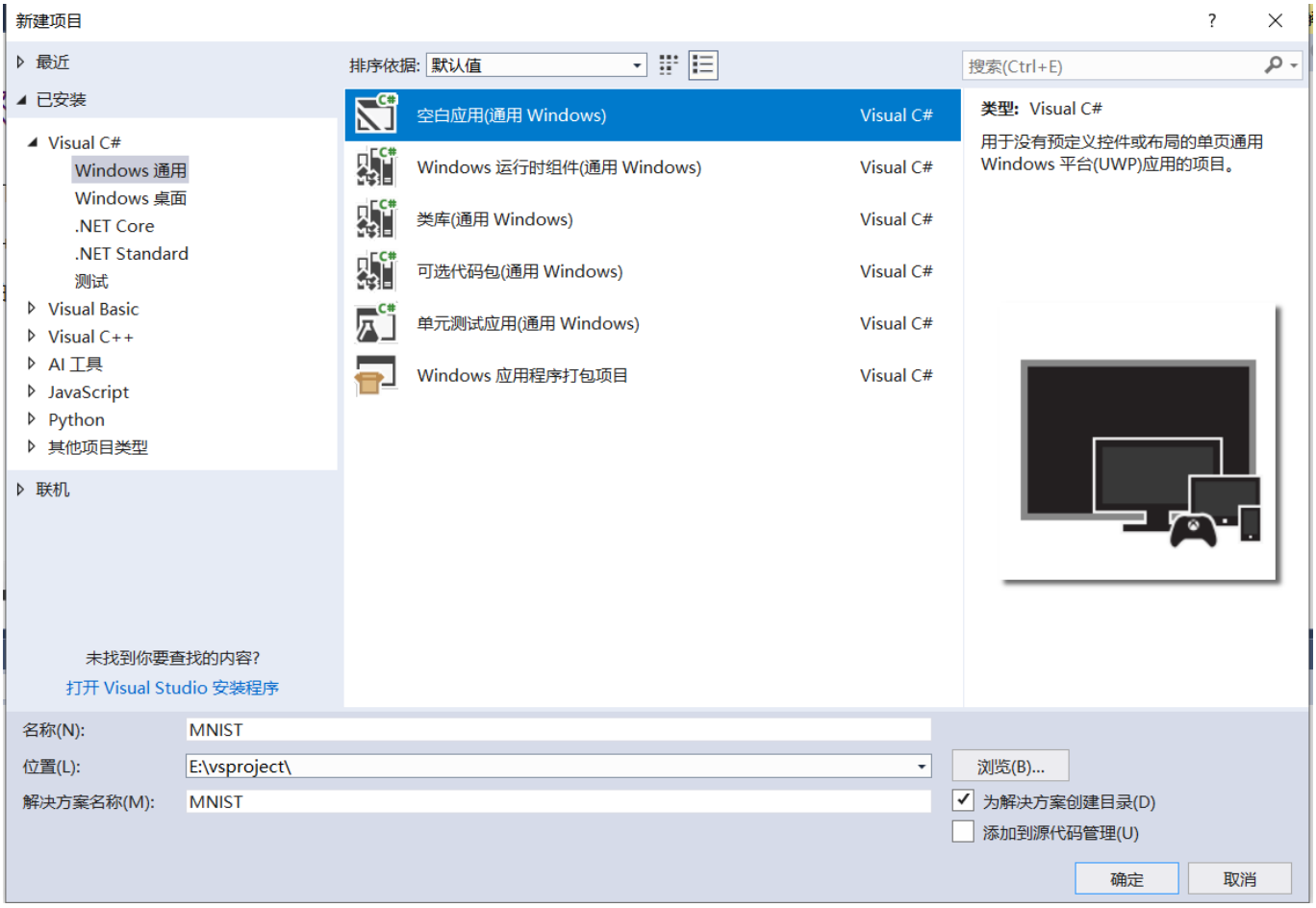
- 一：实现简单的界面，将用户用鼠标或者触屏的输入变成图片。
- 二：将生成的模型包装起来，成为有公开数据接口的类。
- 三：将输入的图片进行规范化，成为数据接口能够使用的格式。
- 四：最后通过模型来推理(inference)出图片应该是哪个数字，并显示出来。

②、具体实现：

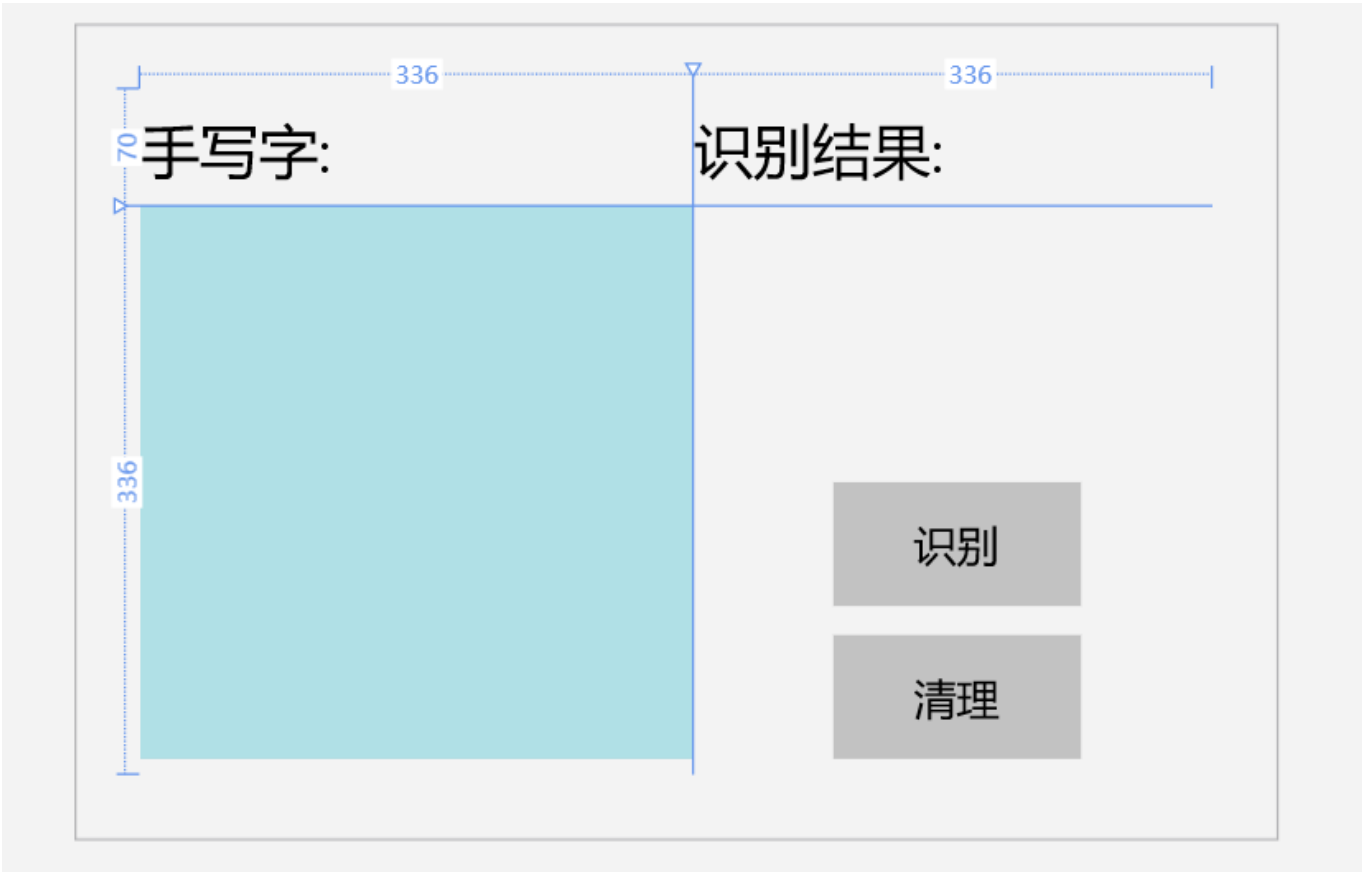
步骤一：获取手写的数字

1、选择文件->新建->项目。

在弹出的窗口里选择Visual C#->Windows窗体应用。



Visual Studio设计窗口的设计图，如下：



其中，设计窗口的部分可以使用代码直接完成，代码如下：

```
<Page
    x:Class="MNIST_Demo.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MNIST_Demo"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" Width="731.195" Height="496">
    <Grid
        x:Name="parentGrid"
        Margin="40">
        <Grid.RowDefinitions>
            <RowDefinition Height="70" />
            <RowDefinition Height="336" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="336" />
            <ColumnDefinition Width="336" />
        </Grid.ColumnDefinitions>

        <TextBlock Text="手写字:"
               FontSize="36"
               VerticalAlignment="Center"
               Grid.Row="0"
               Grid.Column="0" />
        <TextBlock Text="识别结果:"
               FontSize="36"
```

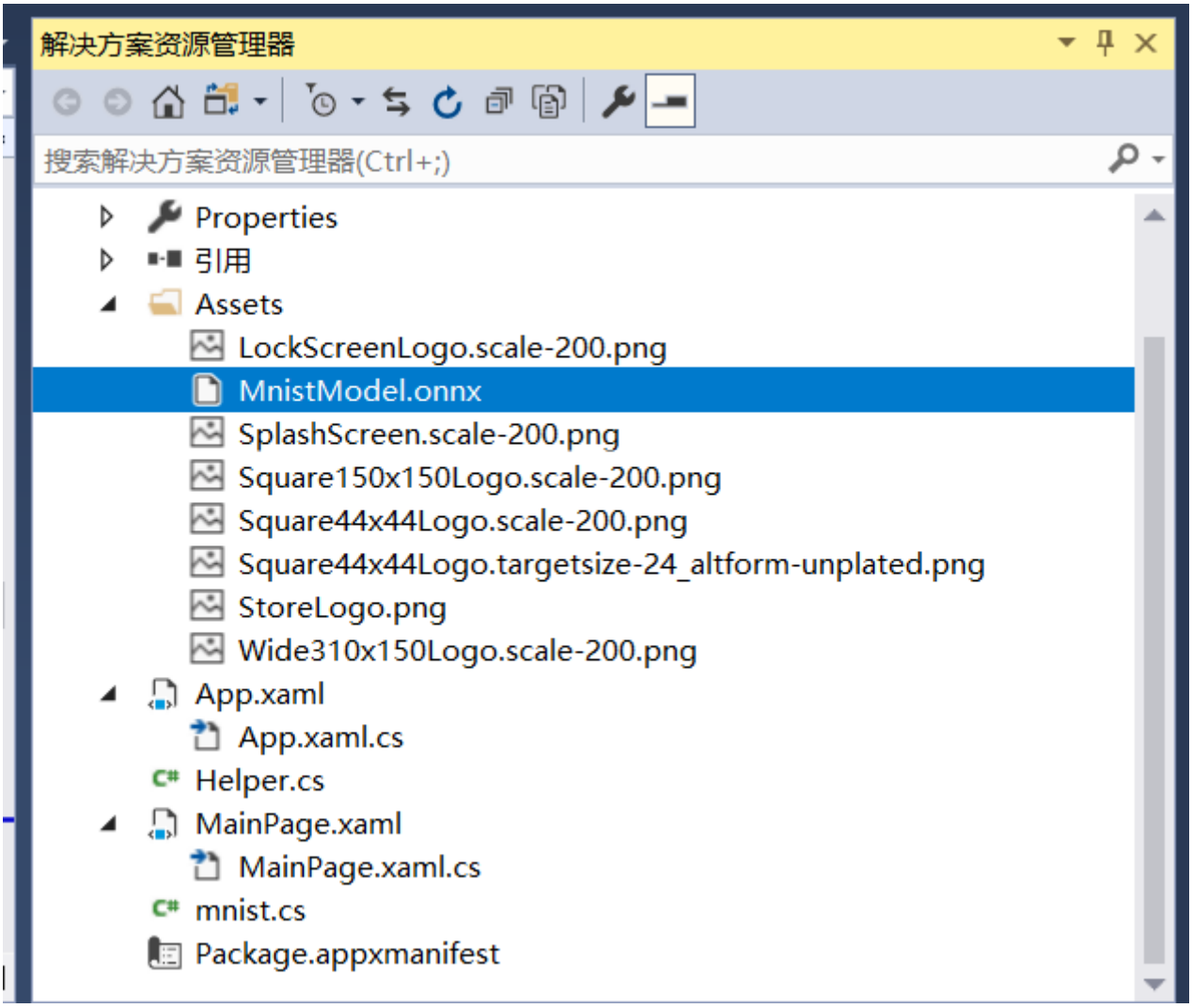
```
            VerticalAlignment="Center"
            Grid.Column="1" Margin="0,0,142,0" />
        <Button
            Name="recognizeButton"
            Content="识别"
            Click="recognizeButton_Click"
            FontSize="26"
            Grid.Column="1"
            Grid.Row="1"
            Height="75"
            Width="150"
            Margin="85,168,0,0"
            VerticalAlignment="Top"/>
        <Button
            Name="clearButton"
            Content="清理"
            Click="clearButton_Click"
            FontSize="26"
            Grid.Column="1"
            Grid.Row="1"
            Height="75"
            Width="150"
            Margin="85,261,0,0"
            VerticalAlignment="Top"/>
        <TextBlock Name="numberLabel"
            FontSize="100"
            Grid.Column="1"
```

```
            Margin="60,0,100,192"

            Text="" VerticalAlignment="Bottom" Grid.Row="1"
SelectionChanged="NumberLabel_SelectionChanged"/>

        <Border BorderThickness="4"

            BorderBrush="Silver"

            Margin="0,0,0,0"

            Grid.Row="1"

            Grid.Column="0"/>



        <Grid Name="inkGrid"

            Background="PowderBlue"
            Grid.Row="1"

            Grid.Column="0">

            <InkCanvas

                    Name="inkCanvas"

                    Height="336"

                    Width="336"/>

        </Grid>

    </Grid>

</Page>
```

③、添加模型

右键单击解决方案资源管理器中的Assets文件夹，然后选择“ 添加” >“ 现有项”。将文件选择器指向ONNX模型的位置，然后单击添加。 该项目现在应该有两个新文件： mnist.onnx-训练的模型，mnist.cs -Windows ML生成

的代码。



- mnist.cs文件中新生成的代码。我们分为三类：

- mnistModel创建机器学习模型表示，在系统默认设备上创建会话，将特定的输入和输出绑定到模型，并异步评估模型。
- mnistInput初始化模型期望的输入类型。在这种情况下，输入需要一个ImageFeatureValue。
- mnistOutput初始化模型将输出的类型。在这种情况下，输出将是TensorFloat类型的名为Plus214_Output_0的列表。

④、加载，绑定和评估模型

a、对于Windows ML应用程序，我们要遵循的模式是：“加载”>“绑定”>“求值”。

加载机器学习模型。 将输入和输出绑定到模型。 评估模型并查看结果。 我们将使用mnist.cs中生成的接口代码来加载，绑定和评估应用程序中的模型。 首先，在MainPage.xaml.cs中，我们实例化模型，输入和输出。将以下成员变量添加到MainPage类：

```
private mnistModel ModelGen;
private mnistInput ModelInput = new mnistInput();
private mnistOutput ModelOutput;
```

b、在LoadModelAsync中，我们将加载模型。

我们使用任何模型的方法（也就是之前这个方法应该叫的MainPage的加载事件，在的OnNavigatedTo覆盖，或之前的任何地方recognizeButton_Click被调用）。该mnistModel类表示MNIST模式并创建系统默认设备上的会话。要加载模型，我们调用CreateFromStreamAsync方法，并传入ONNX文件作为参数。

```
private async Task LoadModelAsync()
{
    // Load a machine learning model
    StorageFile modelFile = await StorageFile.GetFileFromApplicationUriAsync(new
Uri($"ms-appx:///Assets/mnist.onnx"));
    ModelGen = await mnistModel.CreateFromStreamAsync(modelFile as
IRandomAccessStreamReference);
}
```

c、我们要将输入和输出绑定到模型。

生成的代码还包括mnistInput和mnistOutput包装器类。所述mnistInput类表示该模型的预期输入，并且mnistOutput类表示该模型的预期的输出。 要初始化模型的输入对象，请调用mnistInput类构造函数，传入您的应用程序数据，并确保输入数据与模型期望的输入类型匹配。该mnistInput类期待一个ImageFeatureValue，所以我们使用一个辅助方法获取ImageFeatureValue为输入。 使用helper.cs中包含的帮助函数，我们将复制InkCanvas的内容，将其转换为ImageFeatureValue类型，然后将其绑定到我们的模型。

```
private async void recognizeButton_Click(object sender, RoutedEventArgs e)
{
    // Bind model input with contents from InkCanvas
    VideoFrame vf = await helper.GetHandWrittenImage(inkGrid);
    ModelInput.Input3 = ImageFeatureValue.CreateFromVideoFrame(vf);
}
```

对于输出，我们只需使用指定的输入调用EvaluateAsync。输入初始化后，调用模型的EvaluateAsync方法以根据输入数据评估模型。EvaluateAsync将您的输入和输出绑定到模型对象，并在输入上评估模型。 由于模型返回了输出张量，因此我们首先要将其转换为友好的数据类型，然后解析返回的列表以确定哪个数字具有最高的概率并显示该数字。

```
private async void recognizeButton_Click(object sender, RoutedEventArgs e)
{
    // Bind model input with contents from InkCanvas
    VideoFrame vf = await helper.GetHandWrittenImage(inkGrid);
    ModelInput.Input3 = ImageFeatureValue.CreateFromVideoFrame(vf);

    // Evaluate the model
    ModelOutput = await ModelGen.EvaluateAsync(ModelInput);

    // Convert output to datatype
    IReadOnlyList<float> vectorImage =
ModelOutput.Plus214_Output_0.GetAsVectorView();
```

```
    IList<float> imageList = vectorImage.ToList();

    // Query to check for highest probability digit
    var maxIndex = imageList.IndexOf(imageList.Max());

    // Display the results
    numberLabel.Text = maxIndex.ToString();
}
```

⑤、清除InkCanvas

```
private void clearButton_Click(object sender, RoutedEventArgs e)
{
    inkCanvas.InkPresenter.StrokeContainer.Clear();
    numberLabel.Text = "";
}
```

⑥、运行结果：

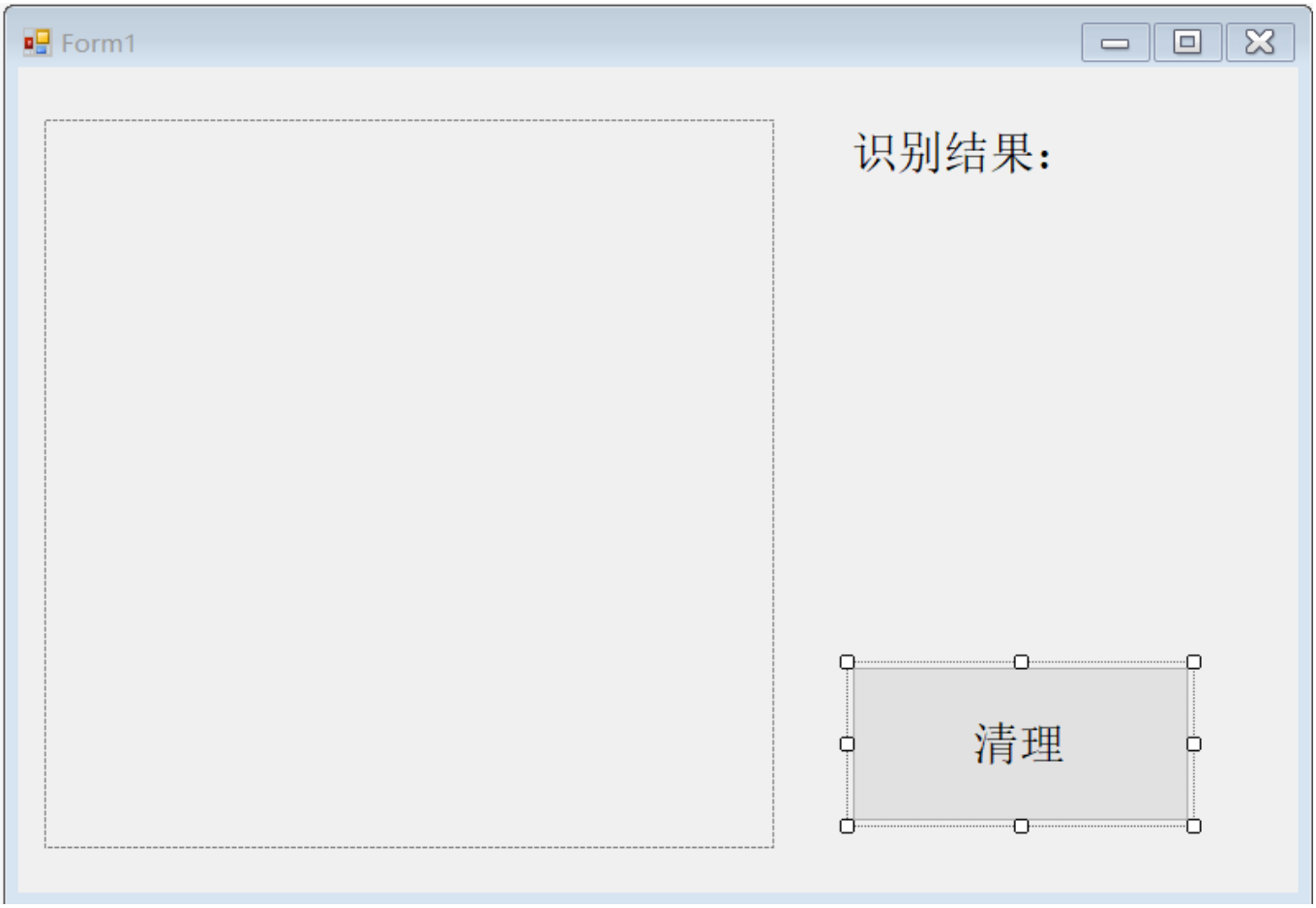

MNIST
手写字:
识别结果:
1
识别
清理


MNIST
手写字:
识别结果:
2
识别
清理

二、基于ONNX Runtime的手写数字推理识别

1、界面设计

创建Windows窗体应用(.NET Framework)项目，这里给项目起名Mnistwf。在解决方案资源管理器中找到Form1.cs，双击，打开界面设计器。从工具箱中向Form中依次拖入控件并调整，最终效果如下图所示：



控件依次是：

- Label控件，将内容改为“ 手写字：”
- pictureBox 写字的面板。
- TextBox控件，为空。
- Button控件，将内容改为“清理”。

2、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，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，使用的模型文件是mnist.onnx。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。

3、添加OnnxRuntime库

微软开源的OnnxRuntime库提供了NuGet包，可以很方便的集成到Visual Studio项目中。 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0.4.0，点击安装，稍等片刻，按提示即可完成安装。 当前NuGet发布的OnnxRuntime库支持x64及x86架构的运行库，建议使用x64的，所以这里将项目的目标架构改为x64。在解决方案上点右键，选择配置管理器。在配置管理器对话框中，将活动解决方案平台切换为x64。如果没有x64，在下拉框中选择新建，按提示新建x64平台。 代码如下：

```
using System;

using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;//用于优化绘制的结果
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Microsoft.ML.OnnxRuntime;
using System.Numerics.Tensors;


namespace Mnistwf
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }


        private Bitmap digitImage;//用来保存手写数字
        private Point startPoint;//用于绘制线段，作为线段的初始端点坐标
        //private Mnist model;//用于识别手写数字
        private const int MnistImageSize = 28;//Mnist模型所需的输入图片大小


        private void Form1_Load(object sender, EventArgs e)
        {
```

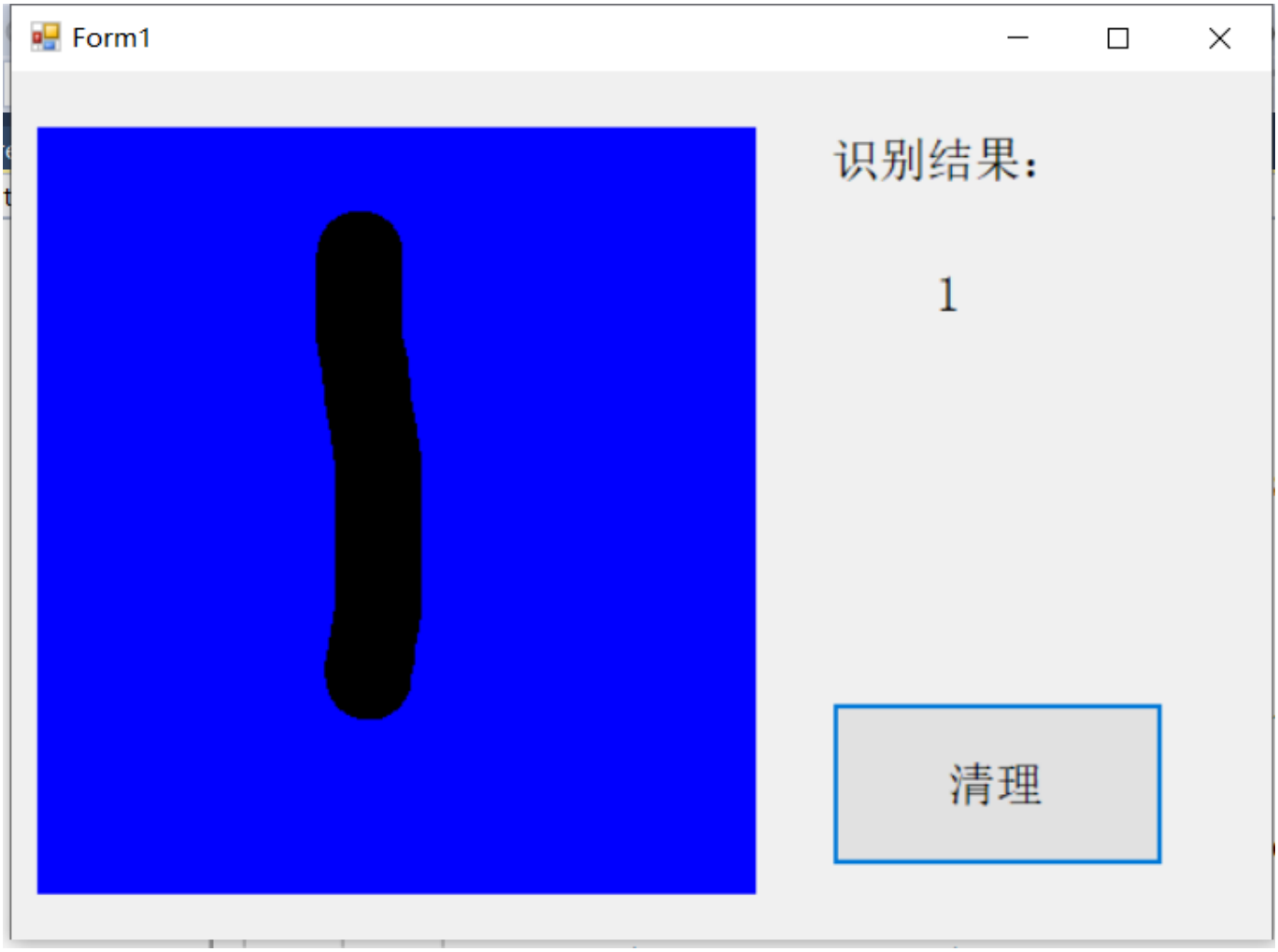
```
            //当窗口加载时，绘制一个白色方框
            //model = new Mnist();
            digitImage = new Bitmap(pictureBox1.Width, pictureBox1.Height);
            Graphics g = Graphics.FromImage(digitImage);
            g.Clear(Color.White);
            pictureBox1.Image = digitImage;
        }

        private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
        {
            //当鼠标左键被按下时，记录下需要绘制的线段的起始坐标
            startPoint = (e.Button == MouseButtons.Left) ? e.Location :
startPoint;
        }

        private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
        {
            //当鼠标在移动，且当前处于绘制状态时，根据鼠标的实时位置与记录的起始坐标绘制
线段，同时更新需要绘制的线段的起始坐标
            if (e.Button == MouseButtons.Left)
            {
                Graphics g = Graphics.FromImage(digitImage);
                Pen myPen = new Pen(Color.Black, 40);
                myPen.StartCap = LineCap.Round;
                myPen.EndCap = LineCap.Round;
                g.DrawLine(myPen, startPoint, e.Location);
                pictureBox1.Image = digitImage;
                g.Dispose();
```

```
            startPoint = e.Location;
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        //当点击清除时，重新绘制一个白色方框，同时清除label1显示的文本
        digitImage = new Bitmap(pictureBox1.Width, pictureBox1.Height);
        Graphics g = Graphics.FromImage(digitImage);
        g.Clear(Color.Blue);//颜色为蓝色
        pictureBox1.Image = digitImage;
        label1.Text = "";
    }

    private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
    {
        //当鼠标左键释放时
        //开始处理图片进行推理
        if (e.Button == MouseButtons.Left)
        {
            Bitmap digitTmp = (Bitmap)digitImage.Clone();//复制digitImage
            //调整图片大小为Mnist模型可接收的大小：28×28
            using (Graphics g = Graphics.FromImage(digitTmp))
            {
                g.InterpolationMode = InterpolationMode.HighQualityBicubic;
                g.DrawImage(digitTmp, 0, 0, MnistImageSize, MnistImageSize);
            }
```

```
                //将图片转为灰阶图，并将图片的像素信息保存在list中
                float[] imageArray = new float[MnistImageSize * MnistImageSize];
                for (int y = 0; y < MnistImageSize; y++)
                {
                    for (int x = 0; x < MnistImageSize; x++)
                    {
                        var color = digitTmp.GetPixel(x, y);
                        var a = (float)(0.5 - (color.R + color.G + color.B) / (3.0
* 255));

                        imageArray[y * MnistImageSize + x] = a;

                    }
                }

                // 设置要加载的模型的路径，跟据需要改为你的模型名称
                string modelPath = AppDomain.CurrentDomain.BaseDirectory +
"mnist.onnx";

                using (var session = new InferenceSession(modelPath))
                {
                    var inputMeta = session.InputMetadata;
                    var container = new List<NamedOnnxValue>();

                    // 用Netron看到需要的输入类型是float32[1, 1, 28, 28]
                    // 第一维None表示可以传入多张图片进行推理
```

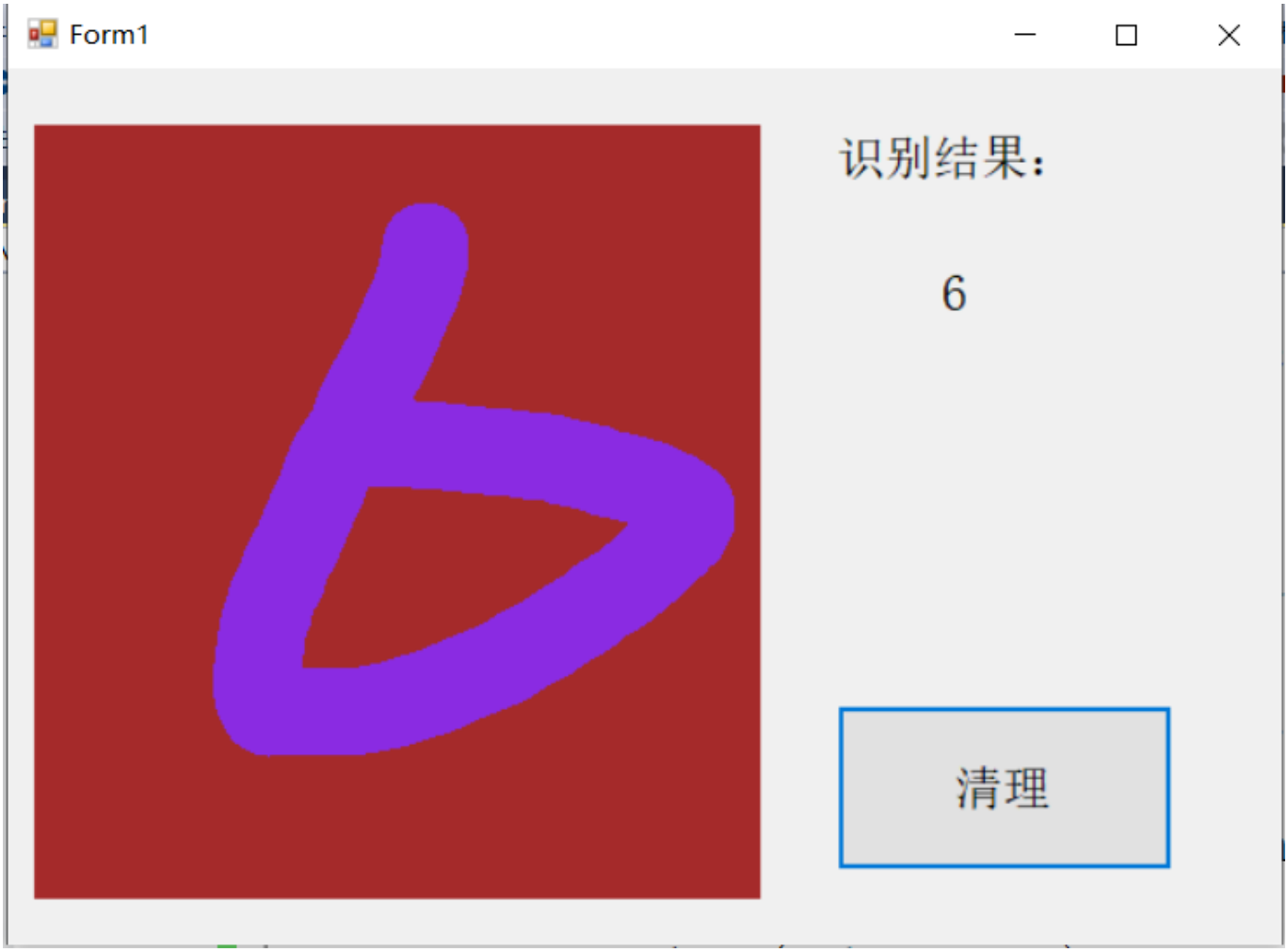
```
                    // 这里只使用一张图片，所以使用的输入数据尺寸为[1, 1, 28, 28]
                    var shape = new int[] { 1, 1, MnistImageSize, MnistImageSize
};
                    var tensor = new DenseTensor<float>(imageArray, shape);

                    // 支持多个输入，对于mnist模型，只需要一个输入，输入的名称是input3
                    container.Add(NamedOnnxValue.CreateFromTensor<float>("Input3",
tensor));

                    // 推理
                    var results = session.Run(container);

                    // 输出结果: Plus214_Output_0
                    IList<float> imageList = results.FirstOrDefault(item =>
item.Name == "Plus214_Output_0").AsTensor<float>().ToList();

                    // Query to check for highest probability digit
                    var maxIndex = imageList.IndexOf(imageList.Max());

                    // Display the results
                    label1.Text = maxIndex.ToString();
                }
            }
        }
        private void pictureBox1_Click(object sender, EventArgs e)
        {
        }
    }
}
```

4、运行结果：




Form1
识别结果：
2
清理


Form1
识别结果:
9
清理



三、总结

通过这次课对学习，我对AI对图像的识别有了一个新的认识，我学习到了如何使用微软的模型来进行图像识别的开发，我通过查阅资料了解到Open Neural Network Exchange (ONNX)是开放生态系统的第一步，它使人工智能开发人员可以在项目的发展过程中选择合适的工具;ONNX为AI models提供了一种开源格式。它定义了一个可以扩展的计算图模型，同时也定义了内置操作符和标准数据类型。在今天调试代码的过程中，我不仅收获了很多，但我也遇到了一些问题。在第一个程序的调试过程中，我能够运行代码，但是弹出窗口后，回到程序中出现错误，通过反复调试，我最终发现是在引用模型的过程中，模型的文件名和我自己在代码中的文件名不一致，最终发生错误。在第二个程序的调试中，我因为没有下载微软的Microsoft.ML.OnnxRuntime包，导致程序一直报错。这两次程序调试过后，我发现自己在调试的过程中不够仔细，在调试方面仍需要加强，另外在代码的理解上我还需要多加练习，多看代码，多练。

在AI学习道路上，我还需要不断努力，虽然这些都是基本的知识点，但是我们学习AI这个领域就要从基础做起，当自己的基础牢固了之后，再继续学习AI的深层领域，才会不会那么吃力。所以，每次我学完一次课就要认真总结学过的知识点，日积月累，我才会不断进步。