计算机综合课程设计报告

——基于卷积神经网络实现图像分类

小 组 姓 名 ： 王欣悦，姚颖舫，黄廷威

指 导 教 师 ： 幸研

学 院 ： 东南大学吴健雄学院

报 告 时 间 ： 2023 年 9 月 15 日

0. 摘要

针对图像分类和识别问题，使用数据集（待填），共包含建筑物、街道、森林、冰川、高山、海洋 6 个类别，通过卷积神经网络深度学习，实现图像分类并提取参数，通过 C++建立神经网络与 Qt 的衔接，并利用 Qt 输出图像及结果。
关键词：深度学习；卷积神经网络；图像分类；特征提取

0. Abstract

Aiming at the problem of image classification and recognition, data sets (to be filled) are used, including 6 categories of buildings, streets, forests, glaciers, mountains and oceans. Image classification and parameter extraction are realized through deep learning of convolutional neural network. The connection between neural network and Qt is established through C++, and the image and results are output using Qt.
Key words: Deep learning; Convolutional neural network; Image classification; Feature extraction

目 录

1. 研究设计

1.1 设计主题

随着近年来深度学习的兴起，计算机视觉所覆盖的诸多领域逐步进入人们的视野中其中图像识别技术相关的研究和应用进展较为突出。图像识别是一种利用计算机对图像进行处理、分析和理解，以识别各种不同模式的目标和对象的技术，是计算机视觉领域的一个主要研究方向，在以图像为主体的智能化数据采集与处理中具有十分重要的作用和影响。使用图像识别技术能够有效地处理特定目标物体的检测和识别（如人脸、手写字符或是商品）、图像的分类标注以及主观图像质量评估等问题。图像分类与识别是神经网络的重要应用，同时也是人脸识别，智能驾驶等技术的来源，计算机视觉(CV)的主要目标。

对于以往的全连接网络(如 ANN)，即使隐藏层只有一层，1000 个神经元，也很容易造成过拟合&计算量，参数量过大的问题。所以利用 ANN 进行图像识别需要的训练时间长，甚至可能达不到学习的目的。此外，网络结构选择没有标准，目前并没有一套统一而完整的理论指导，只能根据算法人员的经验决定。而隐藏层对输入的变量进行各种复杂的非线性组合，导致输出结果难以解释。模型在训练数据上表现良好，但在测试数据上表现不佳，这可能是因为模型过于复杂，导致在训练数据上过度拟合，或者是因为模型对训练数据的噪声过于敏感，导致训练数据中的随机误差被模型放大。

而卷积神经网络则有效避免了以上的问题。卷积神经网络是一种为了处理二维输入数据而特殊设计的多层人工神经网络，网络中的每层都由多个二维平面组成，而每个平面由多个独立的神经元组成，相邻两层的神经元之间互相连接，而处于同一层的神经元之间没有连接。卷积神经网络在卷积层之间减小了数据量，因而擅长处理很多的数据，典型的例子就是图像识别。

1.2 文献综述

图像分类是指根据各自在图像信息中所反映的不同特征，把不同类别的目标区分开来的图像处理方法。它利用计算机对图像进行定量分析，把图像或图像中的每个像元或区域划归为若干个类别中的某一种，以代替人的视觉判读。图像分类的应用场景十分广泛，包括医学图像成像，交通控制等，它是许多其他图像处理的基础，也是机器视觉的基础。图像分类主要有两个类型，有监督分类和无监督分类。有监督分类在训练集中对每个输入的 x（图片），有对应的 y（类别），神经网络需要学习将 x 与 y 相匹配；而无监督分类主要有降维，聚类，异常检测，强化学习等算法，本项目主要探讨有监督分类。

早期的图像分类主要是通过人工标记特征进行分类，将图像转化为颜色，纹理，形状等特征进行特征工程，从而将测量空间转化为特征空间最后转化为类别空间。其中特征空间转化为类别空间的方法主要可以使用 SVM(support vector machine), Random Forest, Decision Tree 等传统的机器学习方法，其优点在于计算速度快，对数据集的要求不高，但是缺点在于人工标注工作量大，而且难以泛化（一组用来识别猫和狗的 SVM 可能难以应用于识别牛和马）。

中期的图像分类将局部特征转化为视觉关键词，比如局部检测子和局部描述子，这样的分类实际上已经有了卷积神经网络的雏形，但是受限于数据集和卷积神经网络的发展问题，这样的图像分类在准确率和泛化程度上仍然存在一些问题。

目前的图像分类主要采用卷积神经网络，最早的卷积神经网络 LeNet5 由 LeCun 和 Bengio 等在 1998 年提出[2]，为 2012 年卷积神经网络的提出奠定了基础，LeNet5 开始用于手写数字识别，由输入层，2 个卷积层，2 个池化层，1 个全连接层和输出层组成。输入为 32*32 的灰度图，C1 采用 6 个 5*5 的卷积核，步长为 1,0 填充；S2 采用 2*2，步长为 2，0 填充的池化；C3 采用 16 个 5*5 的卷积核，步长为 1,0 填充；S4 同样采用 2*2，步长为 2，0 填充的池化;C5 使用 120 个 5*5 的卷积核，事实上可以看作是 Flatten 层，将 2 维的卷积变成了 1 维的数据从而在全连接层输出。这是最早的卷积神经网络结构，也是相对来说最简单，参数量最小的。尽管识别准确率不如后续的网络，但是本次项目由于参数总量的限制（需要转化到 Qt 中读入参数并进行计算，由于事实上在 TensorFlow 中的计算采用了各种优化方法，而这些优化过于繁琐，涉及到了一些利用底层计算机的功能，我们在 C++上难以实现，只能采用朴素的方法，想要在有限时间和运算量（笔记本电脑且没有 GPU）的情况下目前只能采用 Lenet5 的基本模型，事实上我们**本次项目的主要问题就是在运算量，参数量与训练效果之间进行权衡**），尽管如此，本项目在 2k+的测试集中也达到了 80%以上的正确率。

随着时代的发展，更多的神经网络模型迅速诞生，包括 AlexNet[3]、VGGNet、GoogleNet、ResNet[4]以及各种后来发展出来的轻量化网络。事实上 AlexNet（由 5 层卷积层和 3 层全连接层）的出现让深度学习迅速发展起来，而 ResNet 和 GoogleNet 首次在 ImageNet 上的识别程度超过了人类。Resnet 主要由中国科学家何恺明提出，是我们的骄傲。

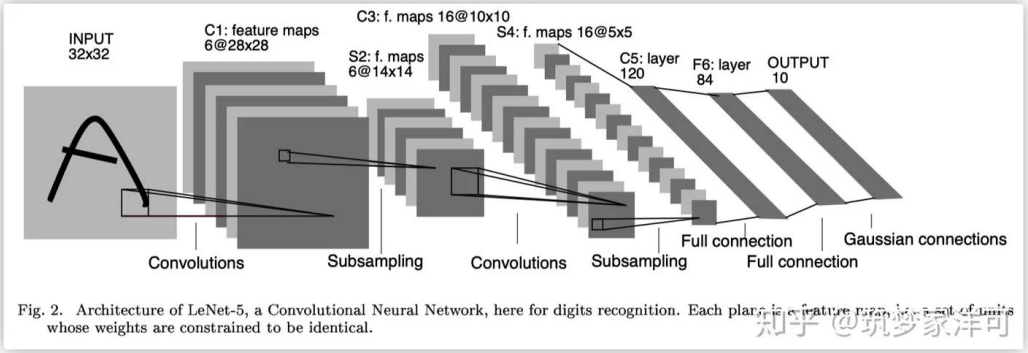


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

图 1LeNet5 框架

1.3 研究思路

卷积神经网络的基本结构分为卷积层、池化层和全连接层。其中，卷积层是卷积神经网络的核心构成部分，其主要作用是提取输入的不同特征。每一个卷积层都由多个卷积单元组成，每一个卷积单元都会学习并提取一种特定的特征。池化层通常设置在卷积层之后，其主要目的是减少特征的维度，降低模型的复杂性，并避免过拟合现象。池化层通过将高维特征映射到低维空间来实现这一目标。全连接层是卷积神经网络的末端部分，其作用是整合前层网络提取的特征，并将这些特征映射到样本标记空间。全连接层中的每一个节点都与前层的节点全部互连，对前层输出的特征进行加权求和，并把结果输入到激活函数，最终完成目标的分类。此外，卷积神经网络中的激活函数用于增加网络的非线性，使得神经网络可以更好地学习和模拟现实中的复杂数据。

一个简单的卷积神经网络模型的结构示意图如图 1 所示，该网络模型由两个卷积层（Ｃ１，Ｃ２）和两个子采样层（Ｓ１，Ｓ２）交替组成。首先，原始输入图像 通过与 3 个可训练的滤波器（或称作卷积核）和可加偏置向量进行卷积运算，在Ｃ１层产生 3 个特征映射图，然后对每个特征映射图的局部区域进行加权平均求和，增加偏置后通过一个非线性激活函数在Ｓ１层得到 3 个新的特征映射图。随后这些特征映射图与Ｃ２层的 3 个可训练的滤波器进行卷积，并进一步通过Ｓ２层后输出 3 个特征映射图。最终Ｓ２层的 3 个输出分别被向量化，然后输入到传统的神经网络中进行训练。这也是本次课程设计所采用的卷积神经网络结构框架。

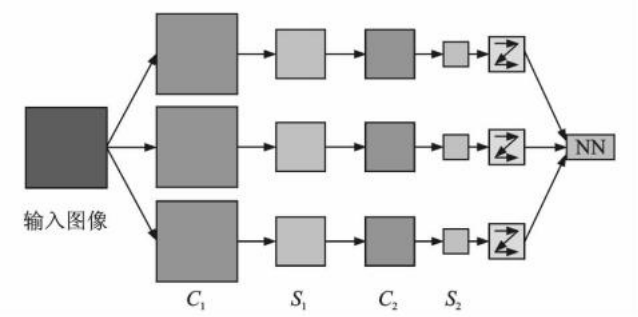


图 1　简化的卷积神经网络结构

Fig. 1　Simplified structure of convolutional neural network

图 1

该论文此后将大致分为四个部分：第一部分介绍本次课程设计的系统框架，其包括 TensorFlow、Qt 的使用介绍；第二部分介绍搭建卷积神经网络提取参数的具体方案和程序的具体编程实现方案、问题解决与面向对象；第三部分介绍 Qt 在该程序中界面展示的设计；第四部分给出部分测试数据与结果，研究其拟合性。

2.系统开发

2.1 模型实现

我们主要使用 TensorFlow2 作为模型的开发工具，开发环境为 Anaconda，使用 Jupyter Notebook 作为编写代码的环境，模型代码的编写语言为 Python 3.10。

在神经网络的模型训练上主要以 LeNet5 为基础，然而我们的输入数据是 150*150*3 的 RGB 信号，输出也是 6 分类的，与 LeNet5 中使用的数据不同，所以我们在此基础上进行了一些改进和适应，改进后的模型如下图所示，训练时间 25 分钟左右（CPU i5-12500，在 GPU 上需要指定 learning rate 初始化一个较小的数值，比如在 GPU V100 上我们指定 Adam(learning_rate = 0.0001)可以得到类似的训练效果并且训练时间被缩短到 1min 以内），共计 50 个 Epoch，Batch_size 为 64，划分 20%的 training example 作为 validation data；采用 Adam 作为优化函数（Adam 可以自适应学习率，如果发现梯度始终向同一方向前进就会适当增加学习率，如果出现了震荡的情况就会自动减小学习率）；使用 Sparse_Categorical_Crossentropy 作为损失函数，该损失函数常用于分类问题；metrics=['accuracy']表明使用精确率来衡量模型的拟合程度，也就是获取模型的准确率。

在具体的模型方面，我们在第二层池化层后加入了 Dropout 层，随机舍弃 25%的神经元，可以有效较少过拟合的发生。事实上我们还尝试了 Batch_Normalization，Batch_Normalization 对输出层的输出数据进行了归一化，但是效果很不好，不仅提高了训练时间，增加了收敛难度，正确率有所下降，Loss_function 的波动明显，查阅资料得出原因可能是 Batch_Normalization 主要是让数据变得“白化”，保持其“独立同分布”的特点，需要用到求平均值和方差的计算，在消除噪声方面比较有用，

而我们的图片中被认为是“噪声”的数据可能也是有用的，而且 Batch_size 不大，可能会因此产生误差。而 Dropout 其实是 Batch_Normalization 的作用之一，在这里效果不错。

在成功完成了模型的训练后，我们将模型的参数以 txt 的形式导出，方便后续 C++代码的读入与实现。

```
In [21]: model = tf.keras.Sequential([
             tf.keras.layers.Conv2D(8, (4, 4), activation = 'relu', input_shape = (150, 150, 3)),
             tf.keras.layers.MaxPooling2D(pool_size = 8, strides = 8),
             tf.keras.layers.Conv2D(16, (2, 2), activation = 'relu'),
             tf.keras.layers.MaxPooling2D(pool_size = 4, strides = 4),
             tf.keras.layers.Dropout(0.25),
             tf.keras.layers.Flatten(),
             tf.keras.layers.Dense(6, activation=tf.nn.softmax)
         ])
         model.summary()
```

图 2 模型结构

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 147, 147, 8)       392

max_pooling2d_10 (MaxPooli   (None, 18, 18, 8)         0
ng2D)

conv2d_11 (Conv2D)           (None, 17, 17, 16)        528

max_pooling2d_11 (MaxPooli   (None, 4, 4, 16)          0
ng2D)

flatten_5 (Flatten)          (None, 256)               0

dense_6 (Dense)              (None, 6)                 1542

=================================================================
Total params: 2462 (9.62 KB)
Trainable params: 2462 (9.62 KB)
```

图 3 模型结构与参数输出

```
In [22]: model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])

In [23]: history = model.fit(train_images, train_labels, batch_size=64, epochs=50, validation_split = 0.2)
```

图 4 模型拟合

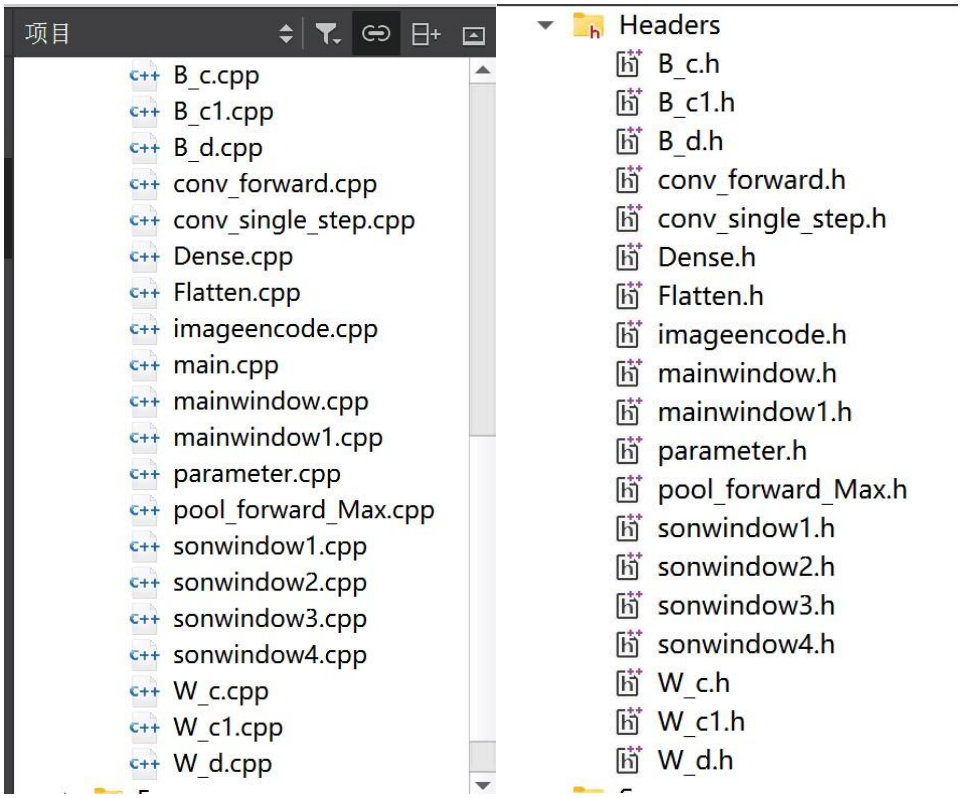
```
with open('RGB.txt', 'w', encoding='utf-8') as f:
    for i in range(len(train_images[0])):
        for j in range(len(train_images[0][i])):
            for k in range(len(train_images[0][i][j])):
                f.write(str(train_images[0][i][j][k]) + ' ')
            f.write('\n')
```

图 5 参数导出

2.2 关键技术

本次项目的关键在于把从 TensorFlow 中提取参数的参数转入 C++中进行实现卷积神经网络的正向传播，查阅资料很难找到前人在这方面作出的成果。

经过充分的讨论，我们的代码主要分为四个部分：图片格式转换部分（imageencode.cpp），卷积神经网络部分（conv_forward.cpp, conv_single_step.cpp, pool_forward_max.cpp, Dense.cpp, Flatten.cpp），读入部分（W_c.cpp, W_c1.cpp, W_d.cpp, b_c.cpp, b_c1.cpp, b_d.cpp），窗口部分（mainwindow.cpp, sonwindow1.cpp, sonwindow2.cpp, sonwindow3.cpp, sonwindow4.cpp），上述部分的 cpp 文件都有对应的类。其中 mainwindow.cpp 储存的是调用卷积神经网络和将参数传到主窗口的部分。



下面展示一下重要的 mainwindow.cpp 和 imageencode.cpp 以及 conv_forward.cpp。

mainwindow.cpp 主要进行的是卷积层-池化层-卷积层-池化层-铺平层-全连接层，并将输出的结果（准确率和输出结果）展示在 mainwindow 的 textlabel 中。

```
    w_c1(tx);
    B_c1(tx);
    w_d(tx);
    B_d(tx);


    file.close();
    int max_f = 150, max_c = 20;
    QVector<QVector<QVector<double>>> Z(max_f, QVector<QVector<double>>(max_f, QVector<double>(max_c)));
    int total_l = 3, max_p = 20;
    QVector<QVector<QVector<QVector<double>>>> Z1
        (total_l, QVector<QVector<QVector<double>>>(max_p, QVector<QVector<double>>
            (max_p, QVector<double>(max_p))));
    QVector<double> Flat((max_unit + 1) * total_class);
    conv_forward(A, W_c[0], b_c[0], Z, 1, 150, 150, 3, 4, 8);
    pool_forward_Max(Z, 8, 8, Z1[0], 147, 147, 8);
    conv_forward(Z1[0], W_c[1], b_c[1], Z1[1], 1, 18, 18, 8, 2, 16);
    pool_forward_Max(Z1[1], 4, 4, Z1[2], 17, 17, 16);
    Flatten(Z1[2], Flat, 16, 4, 4);
    double acc;
    ui->lineEdit->setText(Dense(Flat, W_d, b_d, 256, 6, acc));
    ui->lineEdit_2->setText("Accuracy = " + QString::number(acc));
}
```

Imageencode.cpp 主要进行的是将读入的文件利用 QImage 转化为 150*150 的大小，并转化为 RGB 三个信道，然后将每个信道的数据除以 255（将信号都转化为 0~1 之间有利于缓解梯度爆炸或梯度消失，有利于正则化）。

```
QVector<QVector<QVector<double>>> imageEncode::imageToRGB(QString filename)
{
    QImage img(filename);
    QSize targetSize(150, 150);
    QImage scaledImage = img.scaled(targetSize, Qt::IgnoreAspectRatio, Qt::SmoothTransformation);
    img = scaledImage;
    static QVector<QVector<QVector<double>>> A(150,QVector<QVector<double>>(150, QVector<double>(3)) );
    for(int y = 0; y < img.height();y++)
    {
        for(int x = 0; x < img.width(); x++)
        {
            QRgb rgb = img.pixel(x, y);
            int red = qRed(rgb);
            int green = qGreen(rgb);
            int blue = qBlue(rgb);

            A[y][x][0] = red / 255.0;
            A[y][x][1] = green / 255.0;
            A[y][x][2] = blue / 255.0;
        }
    }
    return A;
}
```

Conv_forward.cpp 进行的是所有 filter，每一个 channel 层的卷积操作，其中的 conv_single_step 类进行的是单个层的卷积操作。

```
void conv_forward(QVector<QVector<QVector<double>>> a_prev,
    QVector<QVector<QVector<QVector<double>>>> W, QVector<double> b,
    QVector<QVector<QVector<double>>> &Z, int stride, int n_H_prev, int n_W_prev,
    int n_C_prev, int f, int n_C)
{
    int n_H = int((n_H_prev - f) / stride) + 1;
    int n_W = int((n_W_prev - f) / stride) + 1;
    int h, w, c, vert_start, horiz_start;
    for(h = 0; h < n_H; h++)
        for (w = 0; w < n_W; w++)
        {
            vert_start = stride * h;
            horiz_start = stride * w;
            QVector<QVector<QVector<double>>> a_slice_prev;
            a_slice_prev.resize(f);
            for (int i = 0; i < f; i++)
            {
                a_slice_prev[i].resize(f);
                for (int j = 0; j < f; j++)
                    a_slice_prev[i][j].resize(n_C_prev);
            }
            for (int i = 0; i < f; i++)
            {
                for (int j = 0; j < f; j++)
                {
                    for (int k = 0; k < n_C_prev; k++)
                    {
                        a_slice_prev[i][j][k] = a_prev[vert_start + i][horiz_start + j][k];
                    }
                }
            }
            QVector<QVector<QVector<double>>> w_slice;
            w_slice.resize(f);
            for (int i = 0; i < f; i++)
            {
                w_slice[i].resize(f);
                for (int j = 0; j < f; j++)
                    w_slice[i][j].resize(n_C_prev);
            }
            for (c = 0; c < n_C; c++)
            {
                for (int i = 0; i < f; i++)
                {
                    for (int j = 0; j < f; j++)
                    {
                        for (int k = 0; k < n_C_prev; k++)
                        {
                            w_slice[i][j][k] = W[i][j][k][c];
                        }
                    }
                }
                Z[h][w][c] = conv_single_step(a_slice_prev, w_slice, b[c], n_C_prev, f);
            }
        }
}
```

2.3 界面搭建

我们使用 Qt 进行界面搭建，Qt 是一个流行的跨平台应用程序开发框架，用于创建图形用户界面（GUI）和应用程序。Qt 提供了大量的类和库，用于开发桌面应用程序、移动应用程序和嵌入式设备应用程序。Qt 最大的优势就是跨平台，开发者可以在 Windows、Linux、macOS 等多个操作系统上开发应用程序，无需针对每个平台编写不同的代码，提高开发效率，减少开发时间和成本。此外，Qt 提供了许多丰富的界面控件，包括标签、按钮、文本框、表格、树形列表等，同时还提供了强大的自定义控件机制，可以满足本次程序设计的需求。

综上所述，Qt 为我们输入图片、输出识别结果提供了良好的交互平台，能够完成课程设计的基本要求。

3. 系统界面

3.1 Qt 使用框架

我们对 Qt 界面的设计是采用多界面结合，具体为两个主界面与四个子界面，其中两个主界面分别为欢迎页和具体应用界面，四个子界面包含在具体应用界面内。两个主界面的连接使用按钮触发。

欢迎界面以背景图片的形式展示本应用可以分类的 6 种物体：街道、海洋、山脉、冰川、森林、建筑。背景图片是通过 label 添加图片的方式添加的。下方有 label 显示提示语和“continue”按钮进入具体应用界面。

具体应用界面主要分为左右两个部分。左部分是图片的提取、分析、处理；右部分是四个按钮，连接四个子界面。左部分将在 3.2 中详细介绍。右部分通过四个按钮连接四个子界面。四个子界面分别展示“数据”“模型”“模型训练效果”“混淆矩阵”相关截图。每个子界面都是通过 label 添加图片的方式，将图片变成整个界面的大小。

3.2 输入界面设计

这里详细介绍 3.1 中提到的具体应用界面的左边部分（图片的提取、分析、处理）。上中下分别分为图片提取、图片处理与展示、图片分析三个部分。

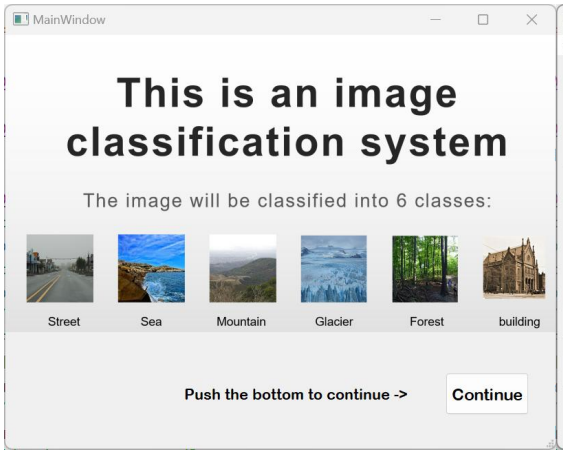
最上方是利用菜单栏的方式，点击菜单栏获取本地图片。其中，imageencode.cpp 将任意大小的图片转换为 150*150 大小的图片，这样，一方面能够分析任意大小的图片，另一方面 150*150 的大小便于后续代码的编写。另外，我们的菜单栏还具备警告功能，如果用户没有选择图片就开始分析，或者点击了菜单栏又关闭（也是没有选择图片），会显示出“请选择一张图片”的提示语。

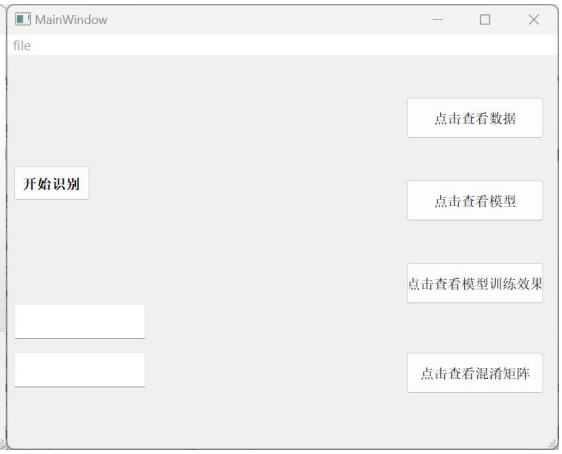
中间部分是一个按钮“开始识别”，用于触发对图片进行细致分析，在按钮右侧是一个方形 label，用于向用户展示他所选择的图片。

下方部分通过两个 label 展示图片的分析结果——具体是什么类的物体，以及分析的准确率如何。

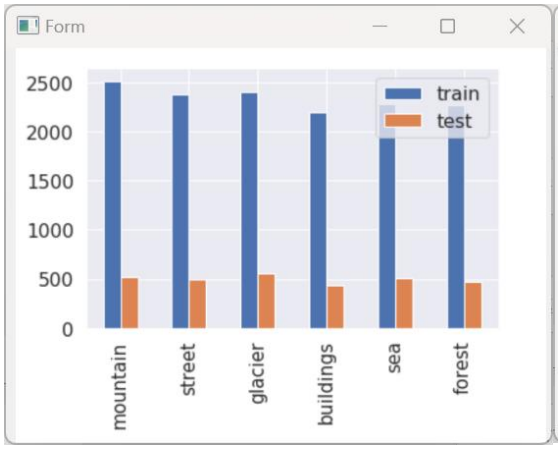
接下来介绍内部的算法。我们总体上做了大规模的封装，将需要读入的参数、各种层的函数、参数读入用的函数都进行了封装。我们在 parameter.cpp 中对各种参数进行了定义，并定义了需要的常量。在 mainwindow.cpp 中进行了具体的参数读入操作。我们封装的函数有 conv_single_step.cpp ，conv_forwad.cpp ,pool_forward_Max.cpp, Flatten.cpp, Dense.cpp 等

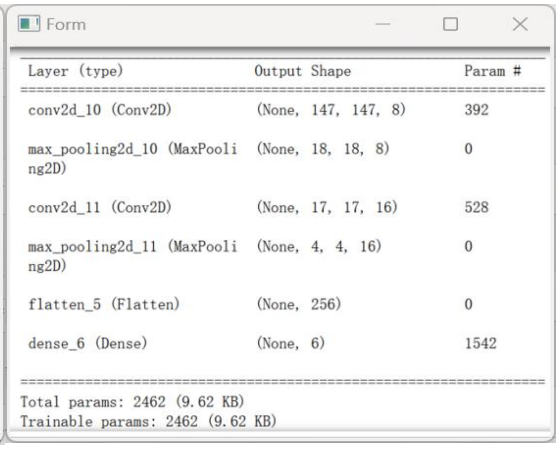
3.3 图形展示界面设计

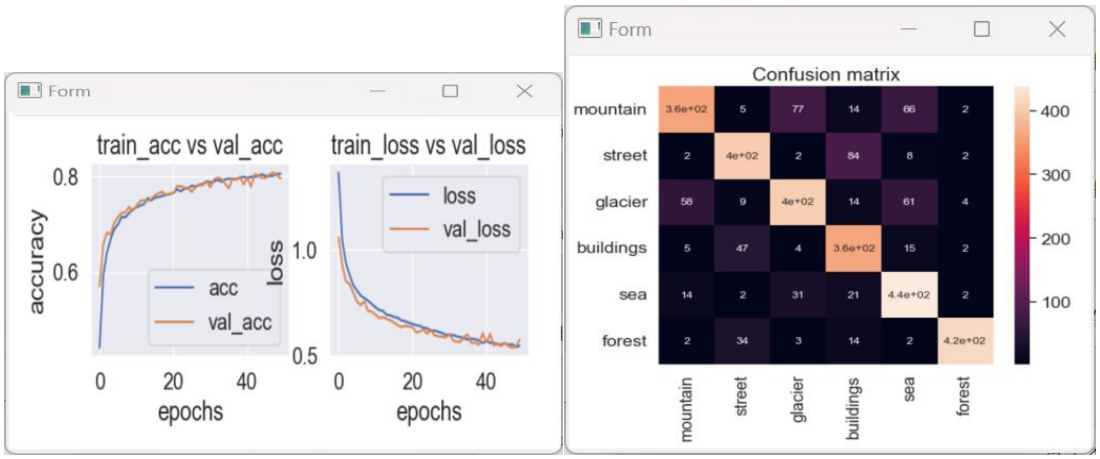




注：点击查看数据，查看模型，查看模型训练效果，查看混淆矩阵后分别显示下面 4 张图片



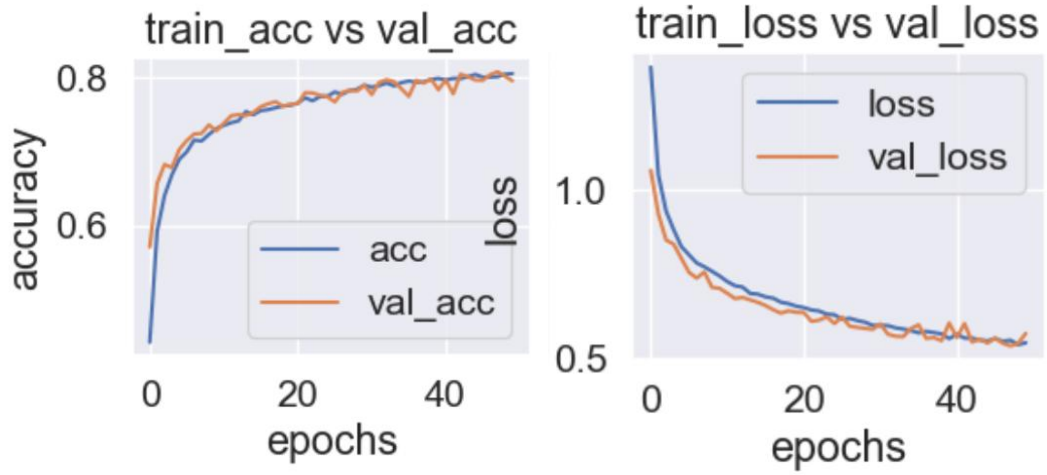




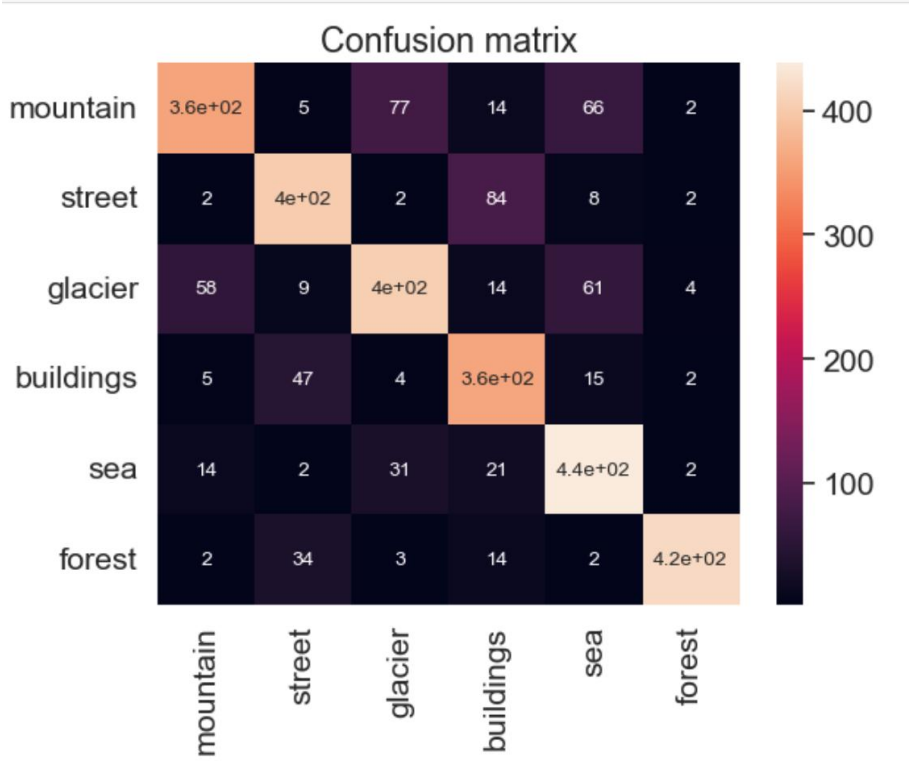
4. 系统测试

4.1 Tensorflow 中的训练效果

如图所示，可以看出在 TensorFlow 中 train_loss 和 validation_loss 同步下降，train_accuracy 和 validation_accuracy 同步上升，train_accuracy 上升较为平缓，而 validation_accuracy 则震荡上升，没有严重的过拟合/欠拟合问题，训练效果较为良好（这可能与我们使用了 Dropout 随机舍弃了 25%的神经网络有关，开始的神经网络在训练到 Epoch=30 的时候就产生了较为严重的过拟合问题，train_accuracy 持续增加，而 validation_accuracy 震荡甚至下降）

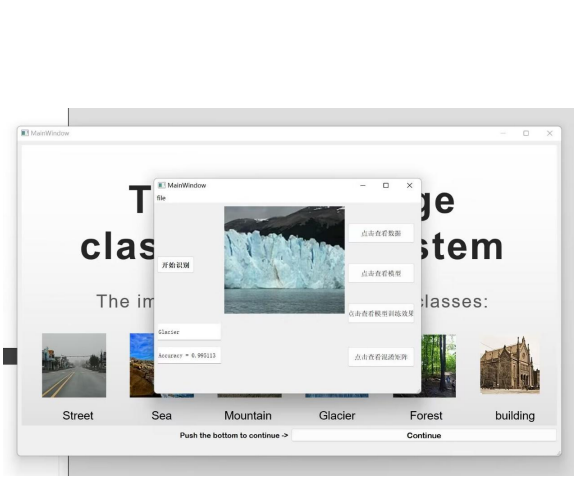


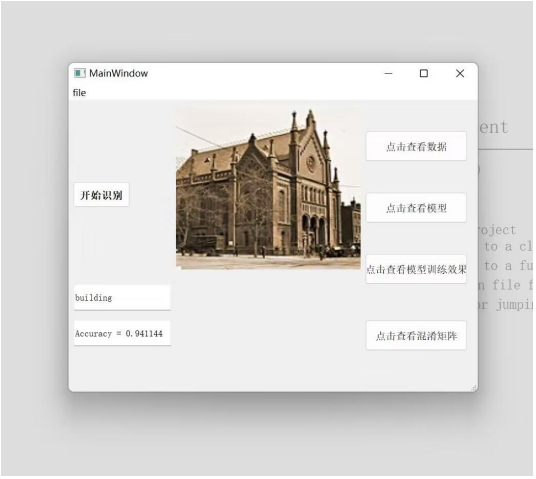
下图是我们使用 tensorflow 模型绘制的混淆矩阵，每一列表示预测数据类别，每一行表示真实数据类别，从中可以看出 forest 的拟合效果不好，而 building 与 street，glacier 与 mountain 容易发生混淆，有一部分 street 被识别成了 forest，这里的原因主要是在真实的数据集中，我们所给的类别信息往往不会独立存在，具体情况详见结尾的结语部分。

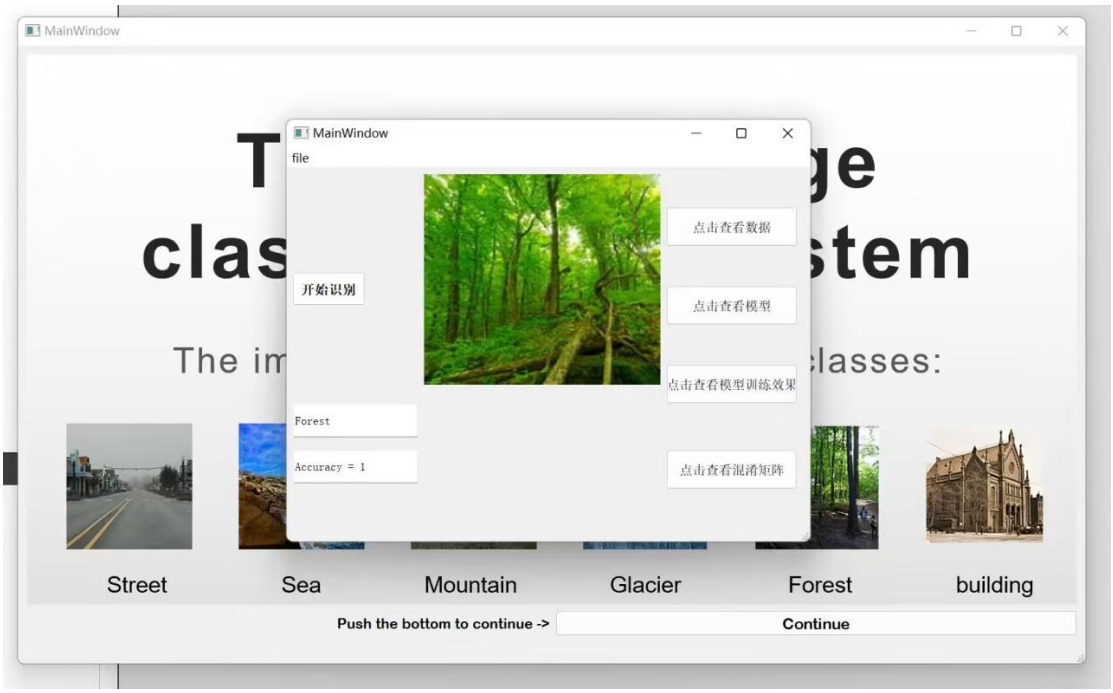


4.2 实际图片的训练效果

4.2.1 大小为 150*150 的训练效果

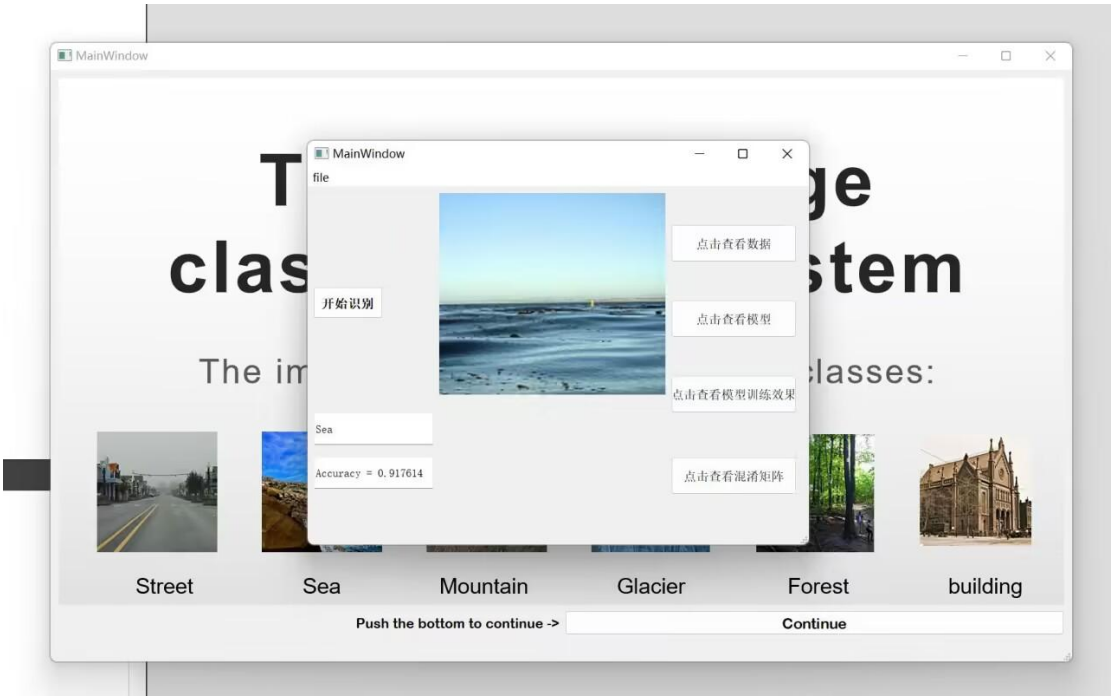


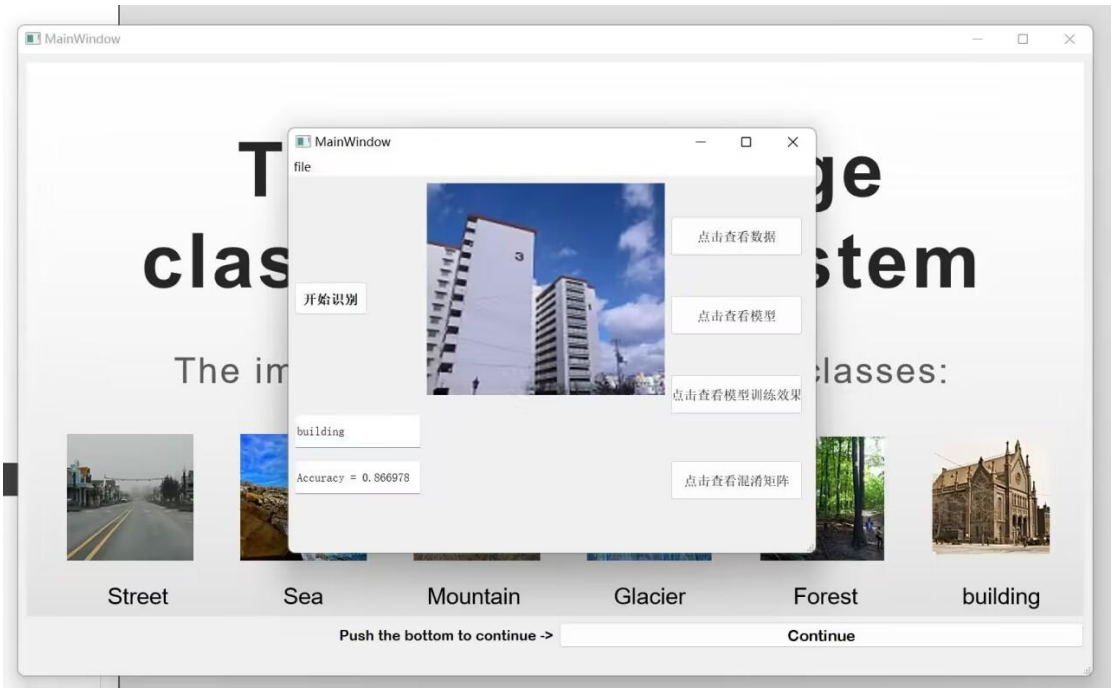


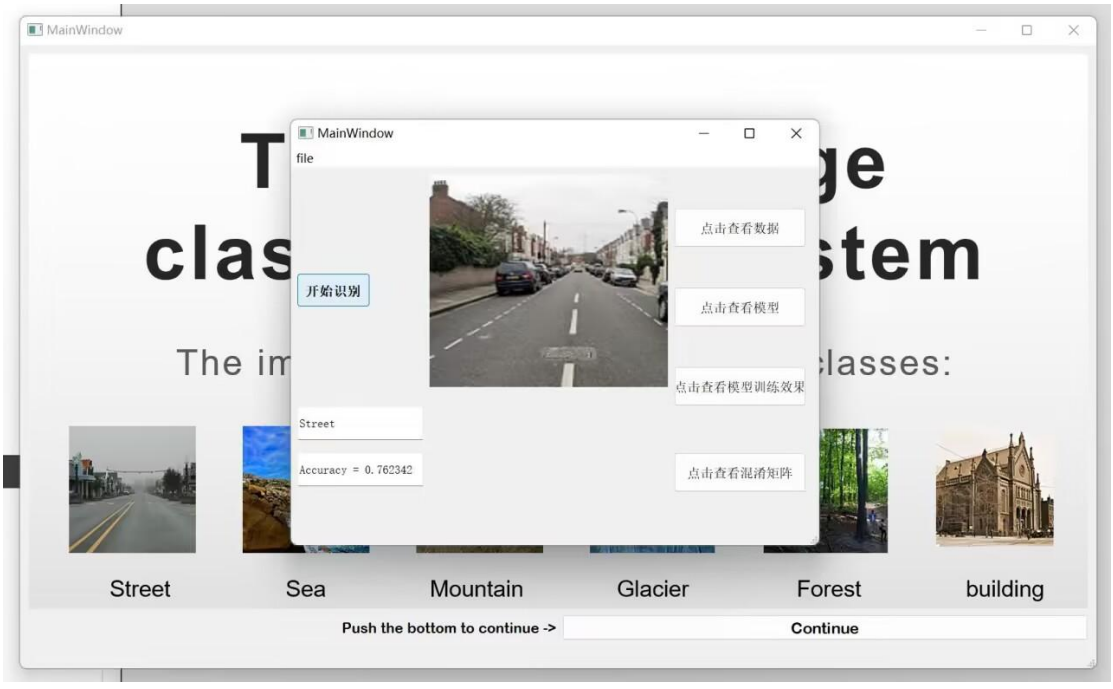


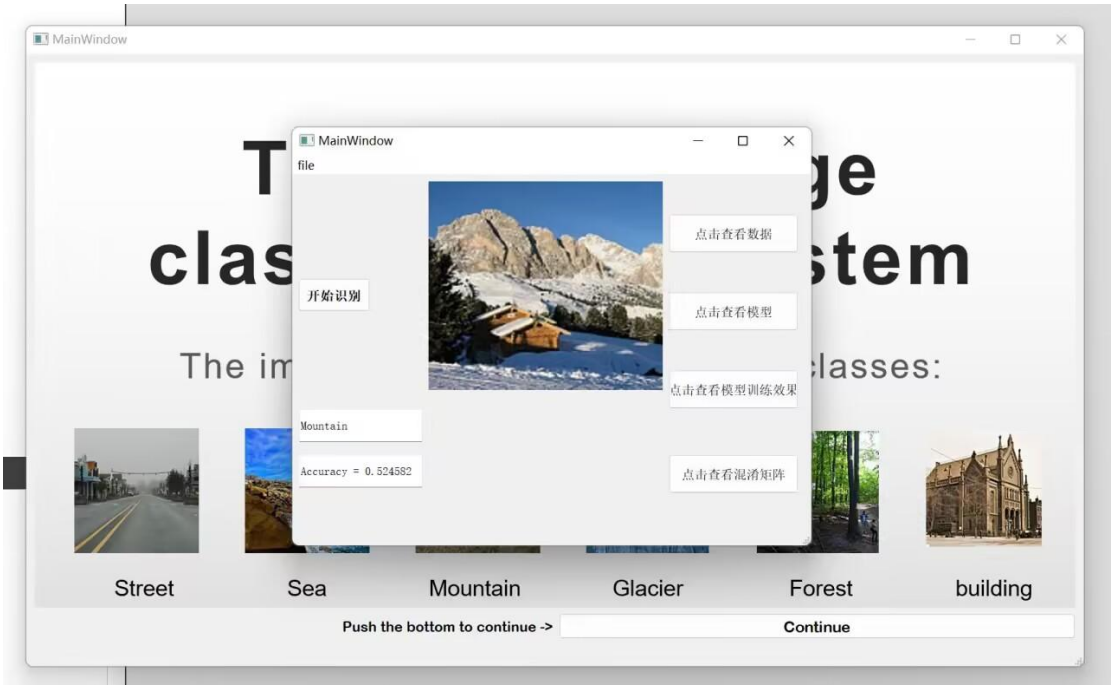
注：这些图片是由数据集提供的专门用来测试的图片，不参与模型的训练

4.2.2 任意大小图片的训练效果




MainWindow
file
开始识别
点击查看数据
点击查看模型
点击查看模型训练效果
building
Accuracy = 0.866978
点击查看混淆矩阵
Street
Sea
Mountain
Glacier
Forest
building
Push the bottom to continue ->
Continue


MainWindow
file
开始识别
点击查看数据
点击查看模型
点击查看模型训练效果
Street
Accuracy = 0.762342
点击查看混淆矩阵
Street
Sea
Mountain
Glacier
Forest
building
Push the bottom to continue ->
Continue



可以看出，任意大小图片的识别效果差一些，可能是因为：

1. 被 150*150 的训练的模型更适合识别正方形的图片；
2. 任意大小图片在大小转换的时候丢失了一些数据；
3. 任意大小图片结构较为复杂，没有数据集中的图片结构简单（只有房子，冰川...）

5. 结语

通过 CNN 人工神经网络的搭建，本课程设计基本完成对（Intel Image Classification）数据库的深度学习并输出有效参数，准确率达到（80）%，可以将任意一张图片转化成 150*150 的像素值并将其区分为建筑物、街道、森林、冰川、高山、海洋中的一个种类，并输出识别准确率。此外（等之后进一步的内容）

因此，我们可以认为，该课程设计的基本内容已经达到预期要求。

此外，在接下来的程序完善过程中我们还将考虑以下目标：

i） Multi-label learning, 从 confusion-matrix 中可以看出，程序在容易将 Glacier-Mountain 混淆，将 Street-Building 混淆，不难得出是因为在某些 training examples 和 test examples 中，冰川是在高山上出现的，高山上偶尔也有冰雪；路边常常有建筑物，建筑物的周边可能也有街道。Multi-label learning 可以同时识别多个 label，在生产生活中的实用性更强。

ii）使用大模型如 Res-net 50, VGG net, 或轻量化模型如 Mobile-net, Efficient-net 等取得更好的拟合效果。关于使用 Res-net 50，以及进行数据增强，本小组的代码详见 https://www.kaggle.com/code/charliestarstar/intel-image-resnet2（最好用 GPU，TPU 实现）

iii）制作更加优美的 UI 界面，主要是自适应窗口大小，进行一些背景设计，做出比较绚丽的 bottom 和 label

6. 致谢

这份结题报告在此暂告收尾。首先要感谢幸研老师，感谢幸老师在我们编写程序过程中给予的帮助与指导，没有他的耐心教导，我们的编写过程会多出很多曲折。最后感谢我们小组的每一位成员：王欣悦，姚颖舫，黄廷威同学，这份程序的最终问世是我们三个人心血的结晶。

7. Reference

[1] A.Krizhevsky,I.Sutskever,G.E.Hinton, ImageNet Classification with Deep Convolutional Neural Networks
[2] LeCun, Y.; Bottou, L.; Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition.Proceedings of the IEEE. 86(11): 2278 - 2324.
[3] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.
[4] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.
[5] Elijah.C, Multi-Label Learning from Single Positive Labels
[6] 周飞燕,金林鹏,董军.卷积神经网络研究综述[J].计算机学报,2017,40(06):1229-1251.
[7] 郑远攀,李广阳,李晔.深度学习在图像识别中的应用研究综述[J].计算机工程与应用,2019,55(12):20-36.

8. 附录

成员分工:
王欣悦：神经网络与参数提取，合作 Qt 与神经网络衔接，PPT,报告
姚颖舫：合作 Qt 与神经网络衔接，报告
黄廷威：划分类，Qt 界面窗口，PPT，报告