Comprehensive Computer Course Design Report

—Image Classification Based on Convolutional Neural Networks

Name :Xinyue Wang, Yinfang Yao, Tingwei Huang

Advisor : Yan Xin

College : Southeast University, Chien-Shiung Wu College

Time : Sep 15, 2023

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

1. Research Design

1.1 Design Abstract

In recent years, with the rise of deep learning, many fields covered by computer vision have gradually come into people's sight, among which the research and application of image recognition technology have made significant progress. Image recognition is a technology that uses computers to process, analyze, and understand images to identify various patterns of objects. It is a major research direction in the field of computer vision and plays a very important role in intelligent data acquisition and processing based on images. Image recognition technology can effectively handle the detection and recognition of specific target objects (such as faces, handwritten characters, or products), image classification and annotation, and subjective image quality assessment. Image classification and recognition are important applications of neural networks and are also the source of technologies such as face recognition and intelligent driving. It is the main goal of computer vision (CV).

For traditional fully connected networks (such as ANN), even if there is only one hidden layer and 1000 neurons, it is easy to cause overfitting and computational problems, as well as excessive parameters. Therefore, using ANN for image recognition requires a long training time and may even fail to achieve the purpose of learning. In addition, there is no standard for selecting the network structure, and there is currently no unified and complete theoretical guidance, which can only be determined based on the experience of algorithm personnel. The hidden layer performs various complex nonlinear combinations of input variables, making the output results difficult to explain. The model performs well on training data but poorly on test data, which may be because the model is too complex, leading to overfitting on the training data, or because the model is too sensitive to noise in the training data, causing random errors in the training data to be amplified by the model.

Convolutional neural networks effectively avoid the above problems. A convolutional neural network is a multi-layer artificial neural network specially designed to process two-dimensional input data. Each layer in the network consists of multiple two-dimensional planes, and each plane consists of multiple independent neurons. The neurons between adjacent layers are interconnected, while there is no connection between neurons in the same layer. Convolutional neural networks reduce the amount of data between convolutional layers, making them good at handling a lot of data, a typical example being image recognition.

1.2 Literature Review

Image classification is an image processing technique that distinguishes different categories of objects based on their respective features reflected in the image information. It utilizes computers to perform quantitative analysis on images, classifying images or individual pixels or regions within images into one of several categories, replacing human visual judgment. Image classification has a wide range of applications, including medical image imaging and traffic control. It serves as the foundation for many other image processing tasks and is fundamental to machine vision.

There are two main types of image classification: supervised classification and unsupervised classification. In supervised classification, for each input x (image) in the training set, there is a corresponding y (category), and the neural network needs to learn to match x with y; while unsupervised classification mainly includes algorithms such as dimensionality reduction, clustering, anomaly detection, and reinforcement learning. This project primarily explores supervised classification.

Early image classification primarily relied on manually labeled features for classification. Images were converted into features such as color, texture, and shape for feature engineering, thereby transforming the measurement space into a feature space and finally into a category space. Methods such as Support Vector Machine (SVM), Random Forest, and Decision Tree, traditional machine learning methods, were commonly used to transform the feature space into the category space. While these methods have the advantages of fast computation speed and low requirements for datasets, they suffer from the drawbacks of a large workload for manual annotation and difficulty in generalization (an SVM used to identify cats and dogs may not be easily applied to identify cows and horses).

In the intermediate stage of image classification, local features were transformed into visual keywords, such as local detectors and local descriptors. This classification already had the embryonic form of convolutional neural networks, but due to limitations in datasets and the development of convolutional neural networks, the accuracy and generalization of such image classification still had some issues.

Currently, convolutional neural networks are primarily used for image classification. The earliest convolutional neural network, LeNet5, was proposed by LeCun and Bengio in 1998 [2], laying the foundation for the emergence of convolutional neural networks in 2012. LeNet5 was initially used for handwritten digit recognition and consisted of an input layer, two convolutional layers, two pooling layers, one fully connected layer, and an output layer. The input was a 32X32 grayscale image. C1 used 6 5X5 convolution kernels with a stride of 1 and 0 padding; S2 used a 2X2 pooling with a stride of 2 and 0 padding; C3 used 16 5X5 convolution kernels with a stride of 1 and 0 padding; S4 also used a 2X2 pooling with a stride of 2 and 0 padding; C5 used 120 5X5 convolution kernels, which can be considered as a Flatten layer, transforming the 2-dimensional convolution into 1-dimensional data, thus outputting in the fully connected layer. This was the earliest convolutional neural network structure and was also the simplest and had the smallest number of parameters. Although the recognition accuracy was not as good as subsequent networks, due to the limitations of the total number of parameters in this project (the parameters needed to be converted and read into Qt for calculation, and in fact, various optimization methods were used in TensorFlow calculations, which were too complex and involved some low-level computer functions that were difficult to implement in C++. We could only use a naive method. To balance computational complexity and training performance within limited time and computing resources (a laptop without a GPU), we could only use the basic model of LeNet5. Despite this, our project achieved an accuracy of over 80% on a test set of 2k+ images.

With the development of time, more neural network models have emerged rapidly, including AlexNet [3], VGGNet, GoogleNet, ResNet [4], and various lightweight networks developed later. In fact, the emergence of AlexNet (consisting of 5 convolutional layers and 3 fully connected layers) has rapidly promoted the development of deep learning, while ResNet and GoogleNet exceeded human performance in image recognition on ImageNet for the first time. ResNet was primarily proposed by Chinese scientist Kaiming He.

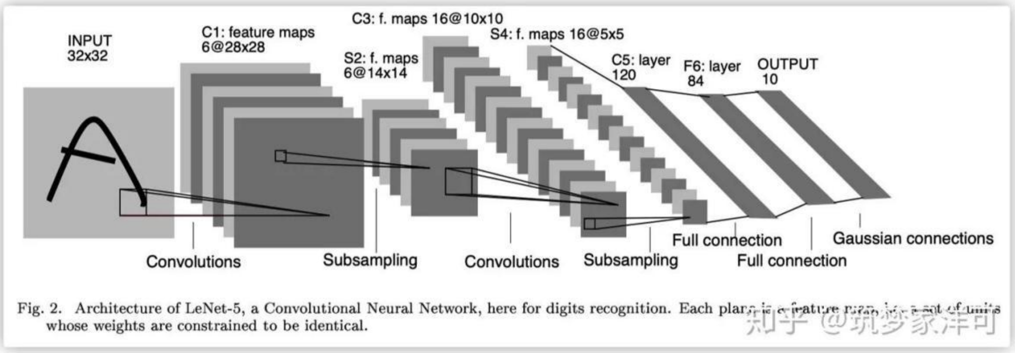


Figure 1: LeNet5 Frame work

1.2 Research ideas

The basic structure of a convolutional neural network (CNN) consists of convolutional layers, pooling layers, and fully connected layers. Among them, the convolutional layer is the core component of the CNN, and its main function is to extract different features from the input. Each convolutional layer is composed of multiple convolutional units, and each convolutional unit learns and extracts a specific feature. The pooling layer is usually placed after the convolutional layer, and its main purpose is to reduce the dimensionality of features, decrease the complexity of the model, and avoid overfitting. The pooling layer achieves this goal by mapping high-dimensional features to a low-dimensional space. The fully connected layer is the end part of the CNN, and its function is to integrate the features extracted by the previous network layers and map these features to the sample label space. Each node in the fully connected layer is fully connected to the nodes of the previous layer, and the weighted sum of the features output from the previous layer is input to the activation function, finally completing the classification of the target. In addition, the activation function in the convolutional neural network is used to increase the nonlinearity of the network, allowing the neural network to better learn and simulate complex data in the real world.

Figure 2 shows the schematic diagram of the structure of a simple convolutional neural network model. This network model consists of two convolutional layers (C1, C2) and two subsampling layers (S1, S2) alternately. First, the original input image is convolved with three trainable filters (or convolution kernels) and an additive bias vector to generate three feature maps in the C1 layer. Then, a weighted average is calculated for the local regions of each feature map, and after adding a bias, a new feature map is obtained in the S1 layer through a nonlinear activation function. Subsequently, these feature maps are convolved with three trainable filters in the C2 layer, and after passing through the S2 layer, three feature maps are output. Finally, the three outputs of the S2 layer are vectorized and then input to a traditional neural network for training. This is also the convolutional neural network structure framework adopted in this course design.

This report will be divided into four parts: the first part introduces the system framework of this course design, including the use of TensorFlow and Qt; the second part introduces the specific scheme for building a convolutional neural network to extract parameters and the specific programming implementation scheme, as well as problem solving and object-oriented programming; the third part introduces the design of the interface display in Qt for this program; and the fourth part provides some test data and results, studying its fitting.

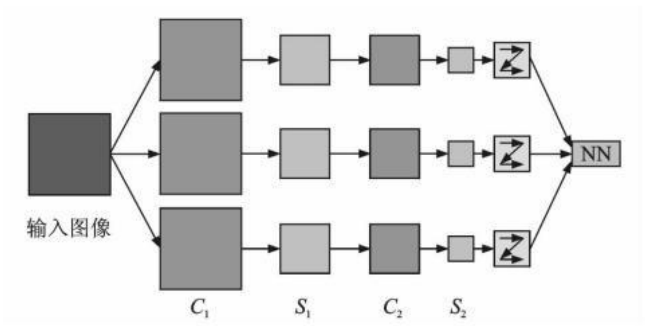


图 1　简化的卷积神经网络结构

Fig. 1　Simplified structure of convolutional neural network

Figure 2

2.System development

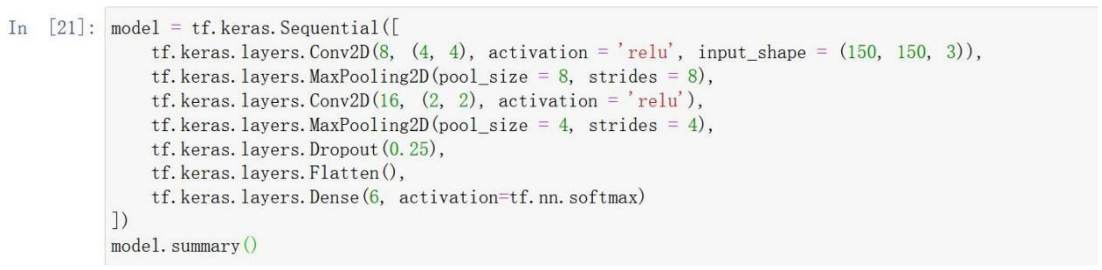
2.1 Model implementation

We primarily used TensorFlow 2 as our model development tool. The development environment was Anaconda, and we used Jupyter Notebook for code writing. The programming language for the model code was Python 3.10.

For neural network model training, we primarily used LeNet5 as a basis. However, our input data is 1501503 RGB signals, and the output is also 6-class, which differs from the data used in LeNet5. Therefore, we made some improvements and adaptations based on it. The improved model is shown in the figure below. The training time is about 25 minutes (CPU i5-12500). On a GPU, a smaller initial learning rate needs to be specified. For example, on a GPU V100, we specified Adam(learning_rate = 0.0001) and obtained similar training results with the training time reduced to less than 1 minute. A total of 50 epochs were used, with a batch size of 64 and 20% of the training examples as validation data. Adam was used as the optimizer (Adam can adaptively adjust the learning rate. If it finds that the gradient is always moving in the same direction, it will appropriately increase the learning rate. If there is an oscillation, it will automatically decrease the learning rate). Sparse_Categorical_Crossentropy was used as the loss function, which is commonly used for classification problems. metrics=['accuracy'] indicates that accuracy is used to measure the goodness of fit of the model, that is, to obtain the accuracy of the model.

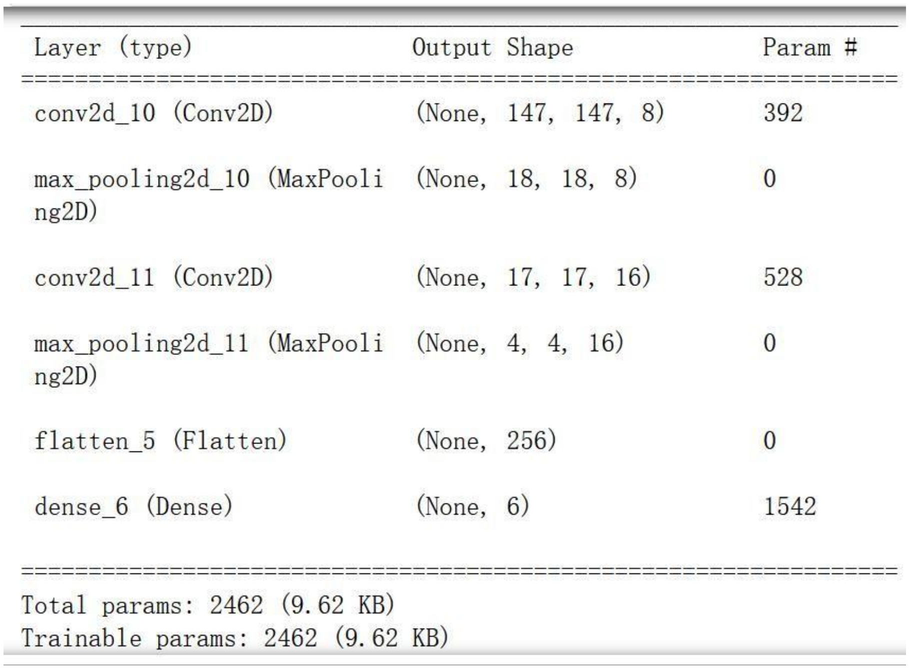
In the specific model, we added a Dropout layer after the second pooling layer, randomly discarding 25% of neurons, which can effectively reduce overfitting. In fact, we also tried Batch Normalization. Batch Normalization normalizes the output data of the output layer, but the effect is not very good. It not only increases the training time and makes convergence more difficult but also reduces the accuracy, and the Loss_function fluctuates significantly. After consulting the literature, we found that the reason may be that Batch Normalization mainly makes the data "whitened" and maintains its "independent and identically distributed" characteristics, which requires the calculation of the mean and variance. It is more useful in eliminating noise, while the data considered as "noise" in our images may also be useful, and the batch size is not large, which may cause errors. Dropout, on the other hand, is actually one of the functions of Batch Normalization, and it works well here.

After successfully training the model, we exported the model parameters in the form of a txt file for subsequent reading and implementation by C++ code.



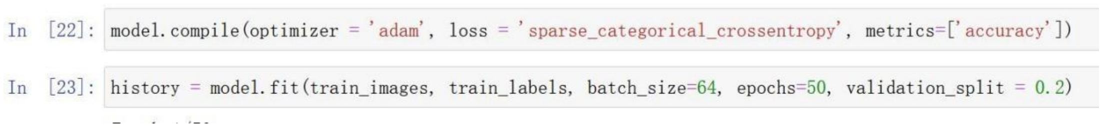
```
In  [21]: model = tf.keras.Sequential([
              tf.keras.layers.Conv2D(8, (4, 4), activation = 'relu', input_shape = (150, 150, 3)),
              tf.keras.layers.MaxPooling2D(pool_size = 8, strides = 8),
              tf.keras.layers.Conv2D(16, (2, 2), activation = 'relu'),
              tf.keras.layers.MaxPooling2D(pool_size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

Figure 3 Code of Model framework



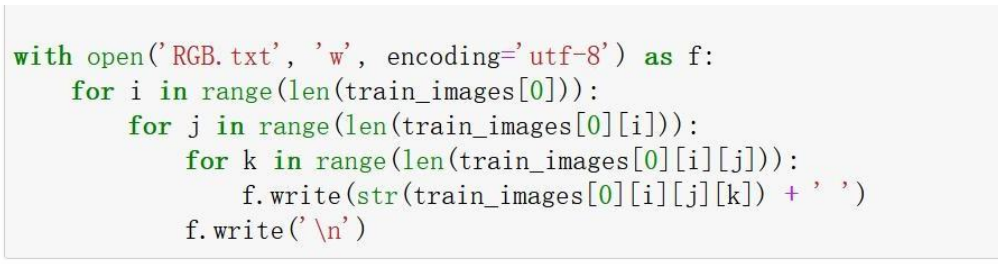
```
Layer (type)                Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)          (None, 147, 147, 8)       392

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)
```

Figure 4 Framework Visualization and Parameters



```
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)
```

Figure 5 Code of Model fit



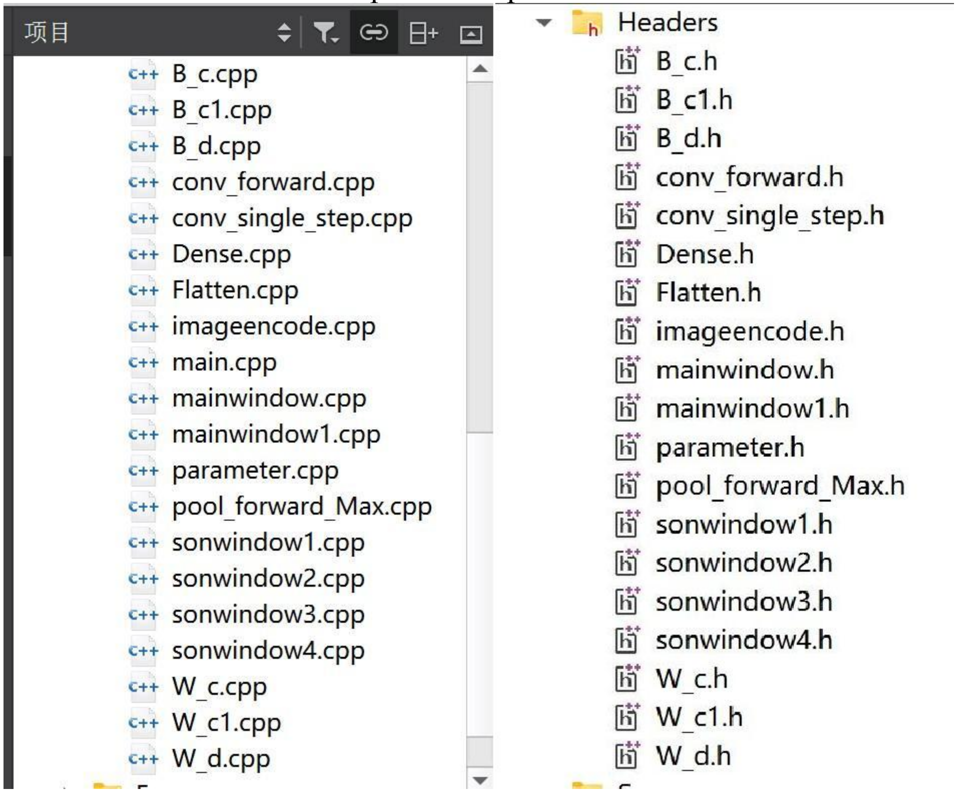
```
with open('RGB.txt', 'w', encoding='utf-8') as f:
    for i in range(len(train_images[0])):
        for j in range(len(train_images[0][i])):
            for k in range(len(train_images[0][i][j])):
                f.write(str(train_images[0][i][j][k]) + ' ')
            f.write('\n')
```

Figure 6: Parameter export

2.2 Key Technology

The key to this project lies in transferring the parameters extracted from TensorFlow to C++ to implement the forward propagation of the convolutional neural network. After extensive research, we found very few previous achievements in this area.

Our code is primarily divided into four parts: image format conversion (imageencode.cpp), convolutional neural network (conv_forward.cpp, conv_single_step.cpp, pool_forward_max.cpp, Dense.cpp, Flatten.cpp), reading (W_c.cpp, W_c1.cpp, W_d.cpp, b_c.cpp, b_c1.cpp, b_d.cpp), and window (mainwindow.cpp, sonwindow1.cpp, sonwindow2.cpp, sonwindow3.cpp, sonwindow4.cpp). Each of the above cpp files has a corresponding class. Among them, mainwindow.cpp stores the part that calls the convolutional neural network and passes the parameters to the main window.



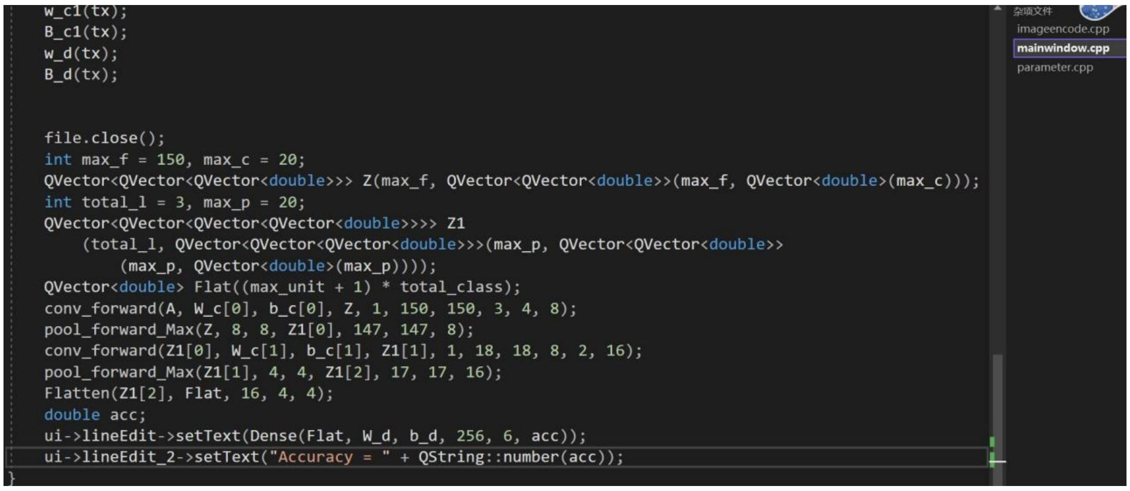
项目

B_c.cpp
B_c1.cpp
B_d.cpp
conv_forward.cpp
conv_single_step.cpp
Dense.cpp
Flatten.cpp
imageencode.cpp
main.cpp
mainwindow.cpp
mainwindow1.cpp
parameter.cpp
pool_forward_Max.cpp
sonwindow1.cpp
sonwindow2.cpp
sonwindow3.cpp
sonwindow4.cpp
W_c.cpp
W_c1.cpp
W_d.cpp

Headers
B_c.h
B_c1.h
B_d.h
conv_forward.h
conv_single_step.h
Dense.h
Flatten.h
imageencode.h
mainwindow.h
mainwindow1.h
parameter.h
pool_forward_Max.h
sonwindow1.h
sonwindow2.h
sonwindow3.h
sonwindow4.h
W_c.h
W_c1.h
W_d.h

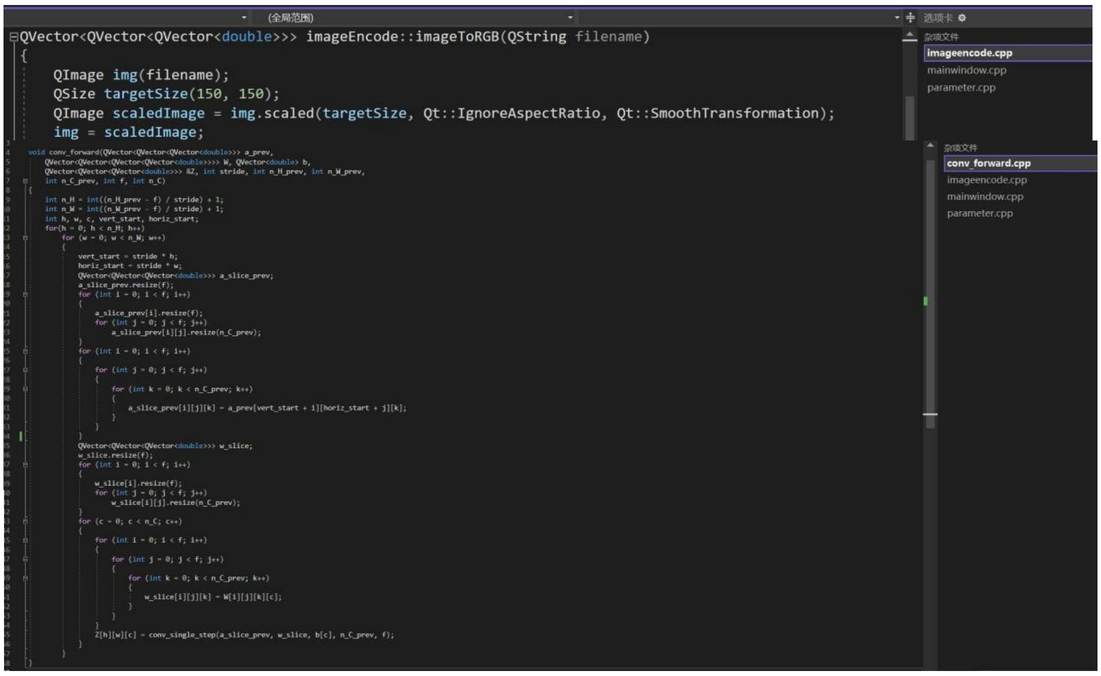
The following demonstrates the important mainwindow.cpp, imageencode.cpp, and conv_forward.cpp.

mainwindow.cpp mainly performs the following operations: convolutional layer - pooling layer - convolutional layer - pooling layer - flattening layer - fully connected layer, and displays the output results (accuracy and output) in the textlabel of the main window.



```
    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 mainly converts the read file into a 150*150 size using QImage and converts it into three RGB channels. Then, the data of each channel is divided by 255 (converting all signals to between 0 and 1 helps alleviate gradient explosion or vanishing gradient and is beneficial for regularization).



```
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;
```

```
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);
            }
        }
}
```

Conv_forward.cpp performs the convolution operation for all filters and each channel layer. The conv_single_step class performs the convolution operation of a single layer.

2.3 Interface Construction

We utilized Qt to construct the user interface. Qt is a popular cross-platform application development framework used to create graphical user interfaces (GUIs) and applications. Qt provides a vast array of classes and libraries for developing desktop applications, mobile applications, and embedded device applications. Qt's greatest advantage is its cross-platform nature, allowing developers to create applications on multiple operating systems such as Windows, Linux, and macOS without the need to write different code for each platform, thereby increasing development efficiency and reducing development time and costs. Additionally, Qt offers a rich set of interface controls, including labels, buttons, text boxes, tables, tree lists, and a powerful custom control mechanism, which can meet the requirements of this program design.

In summary, Qt provides an excellent interactive platform for us to input images and output recognition results, fulfilling the basic requirements of our course design.

3. System interface

3.1 Qt Framework Usage

Our Qt interface design adopts a multi-interface combination, specifically two main interfaces and four sub-interfaces. The two main interfaces are the welcome page and the specific application interface, while the four sub-interfaces are contained within the specific application interface. The two main interfaces are connected by button triggers.

The welcome page displays six types of objects that the application can classify (streets, oceans, mountains, glaciers, forests, and buildings) in the form of a background image. The background image is added using a label. Below, there is a label displaying a prompt and a "continue" button to enter the specific application interface.

The specific application interface is mainly divided into two parts: left and right. The left part is for image extraction, analysis, and processing; the right part contains four buttons that connect to four sub-interfaces. The left part will be detailed in section 3.2. The right part connects to four sub-interfaces through four buttons. The four sub-interfaces display screenshots related to "data," "model," "model training results," and "confusion matrix," respectively. Each sub-interface displays an image as the entire interface size by adding an image to a label

3.2 Input Interface Design

Here we detail the interface designed by Section 3.1 the left part of the specific application interface (image extraction, analysis, and processing). The top, middle, and bottom parts are divided into image extraction, image processing and display, and image analysis, respectively.

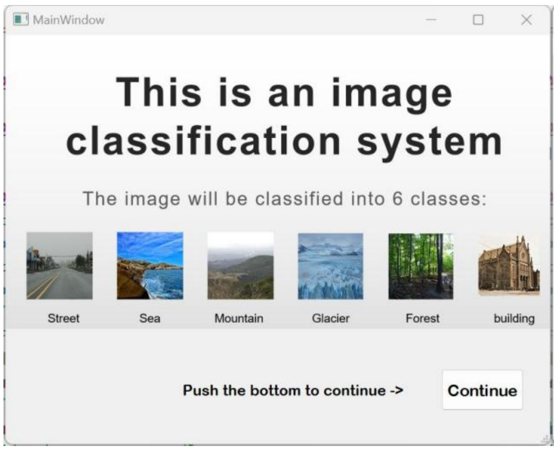
At the top, a menu bar is used to obtain local images by clicking on the menu bar. Imageencode.cpp converts images of any size to a 150X150 size, allowing the analysis of images of any size and making it easier to write subsequent code with a size of 150X150. In addition, our menu bar has a warning function. If the user does not select an image before starting the analysis or clicks the menu bar and closes it (without selecting an image), a prompt "Please select an image" will be displayed.

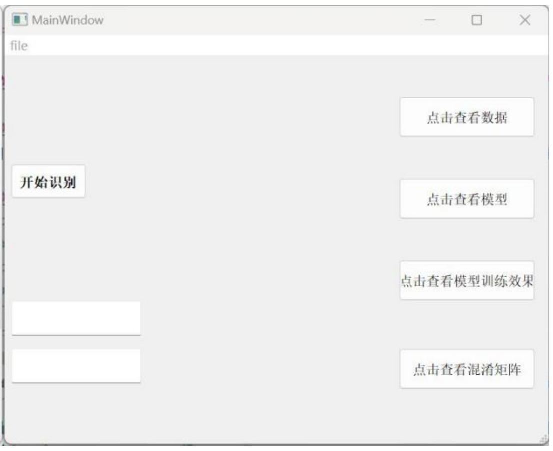
In the middle part, there is a button labeled "Start Recognition" to trigger a detailed analysis of the image. On the right side of the button is a square label used to display the image selected by the user.

The bottom part displays the analysis results of the image through two labels - the specific type of object and the analysis accuracy.

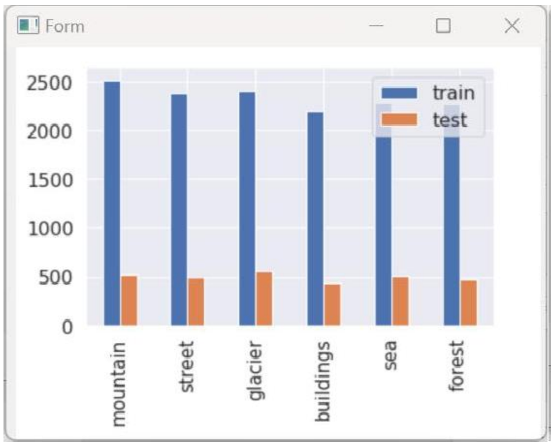
Next, we introduce the internal algorithm. We have done large-scale encapsulation, encapsulating the parameters to be read, functions of various layers, and functions for reading parameters. We define various parameters in parameter.cpp and define the required constants. The specific parameter reading operation is performed in mainwindow.cpp. The functions we encapsulated include conv_single_step.cpp, conv_forwad.cpp, pool_forward_Max.cpp, Flatten.cpp, and Dense.cpp.

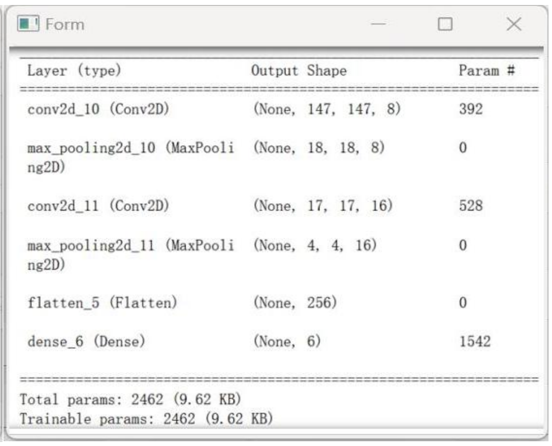
3.3 GUI design


MainWindow
This is an image classification system
The image will be classified into 6 classes:
Street
Sea
Mountain
Glacier
Forest
building
Push the bottom to continue ->
Continue

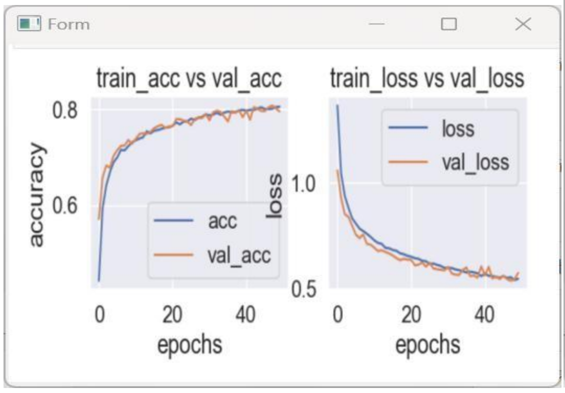


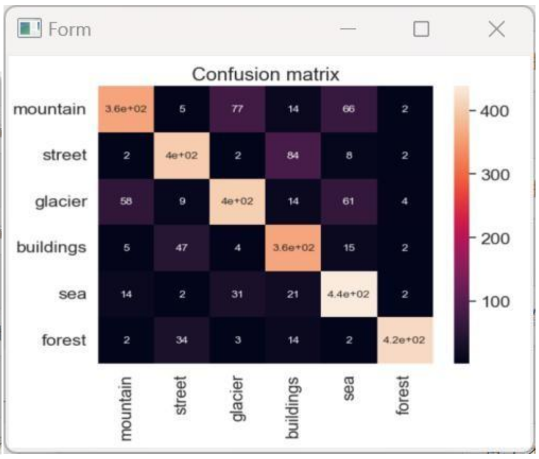
Note: Click to view data, view model, view model training effect, and after viewing the confusion matrix, the following 4 images will be displayed respectively





```
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)
```

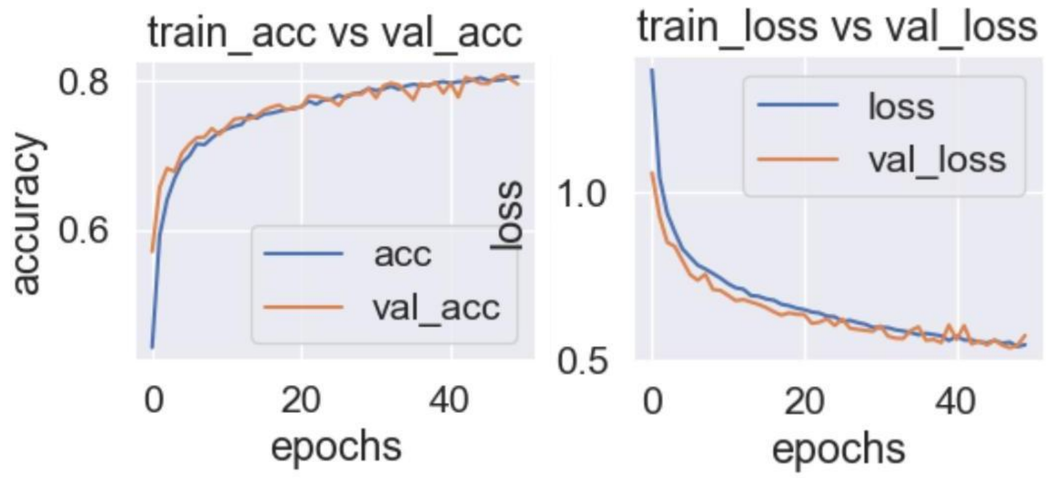




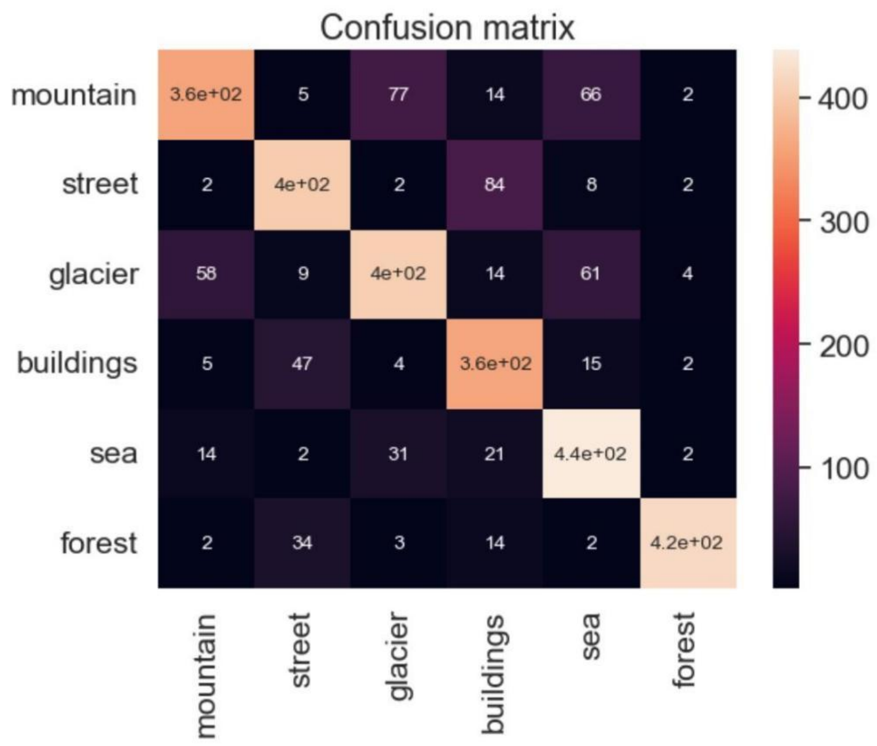
4. System Test

4.1 Model Training effect in Tensorflow

As shown in the figure, both the train_loss and validation_loss decrease synchronously in TensorFlow, while the train_accuracy and validation_accuracy increase synchronously. The train_accuracy increases relatively slowly, while the validation_accuracy fluctuates upward. There is no serious overfitting or underfitting problem, and the training effect is relatively good (this may be related to the fact that we used Dropout to randomly drop 25% of the neurons. At the beginning of the training, when the neural network reached Epoch=30, there was a more serious overfitting problem, where the train_accuracy continued to increase, while the validation_accuracy fluctuated or even decreased).

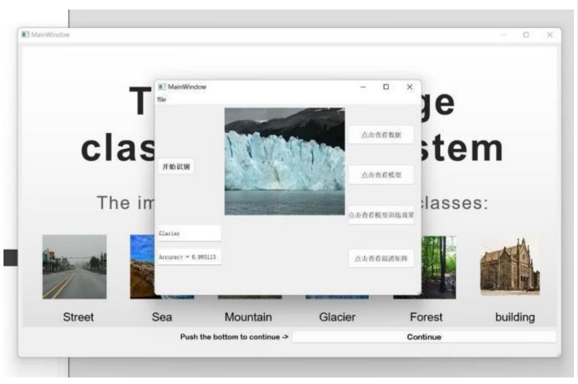


The following figure is the confusion matrix plotted using the TensorFlow model. Each column represents the predicted data class, and each row represents the true data class. It can be seen that the fitting effect of "forest" is not good, and "building" is easily confused with "street", and "glacier" is easily confused with "mountain". Some "street" is identified as "forest". The main reason for this is that in the real dataset, the class information we provide often does not exist independently. For more details, please refer to the conclusion section at the end.

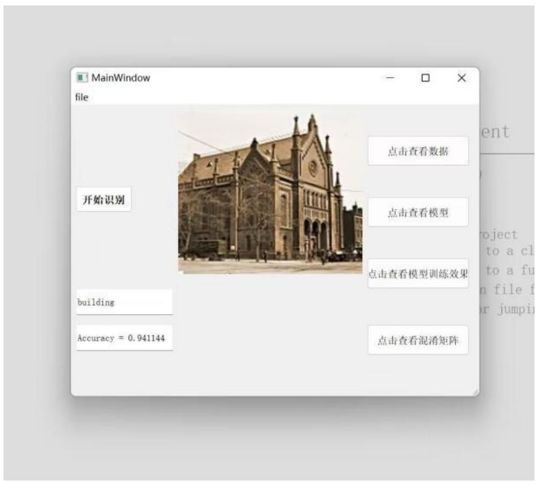


4.2 Training effect of actual images

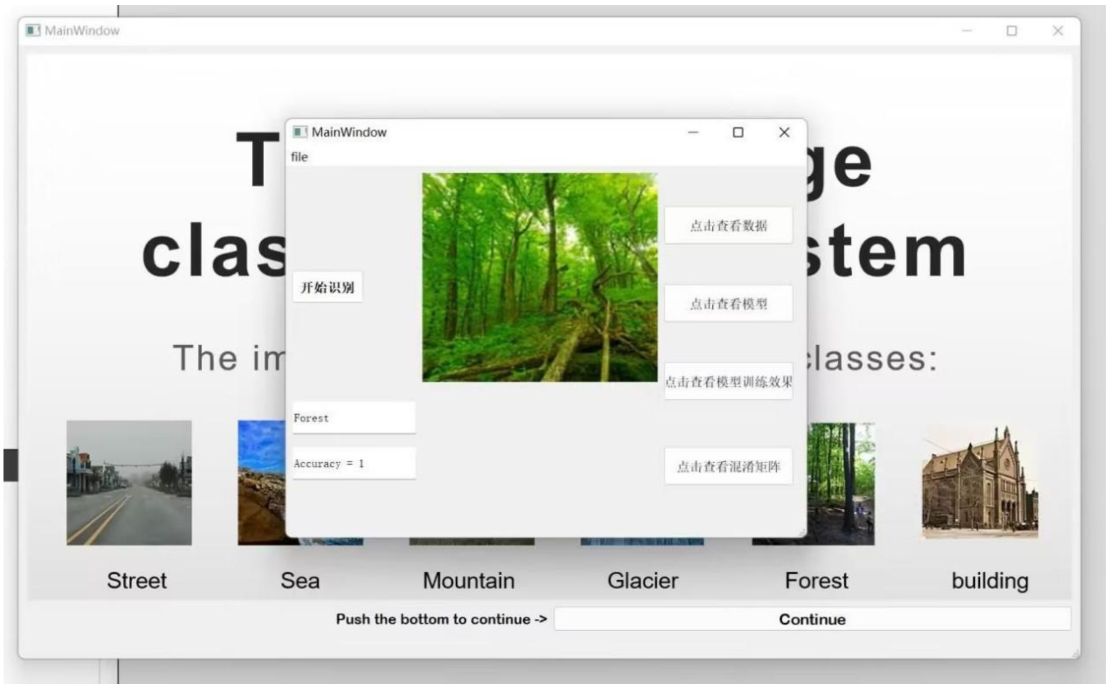
4.2.1 Image with 150*150



```
Glacier
Accuracy = 0.995113
```

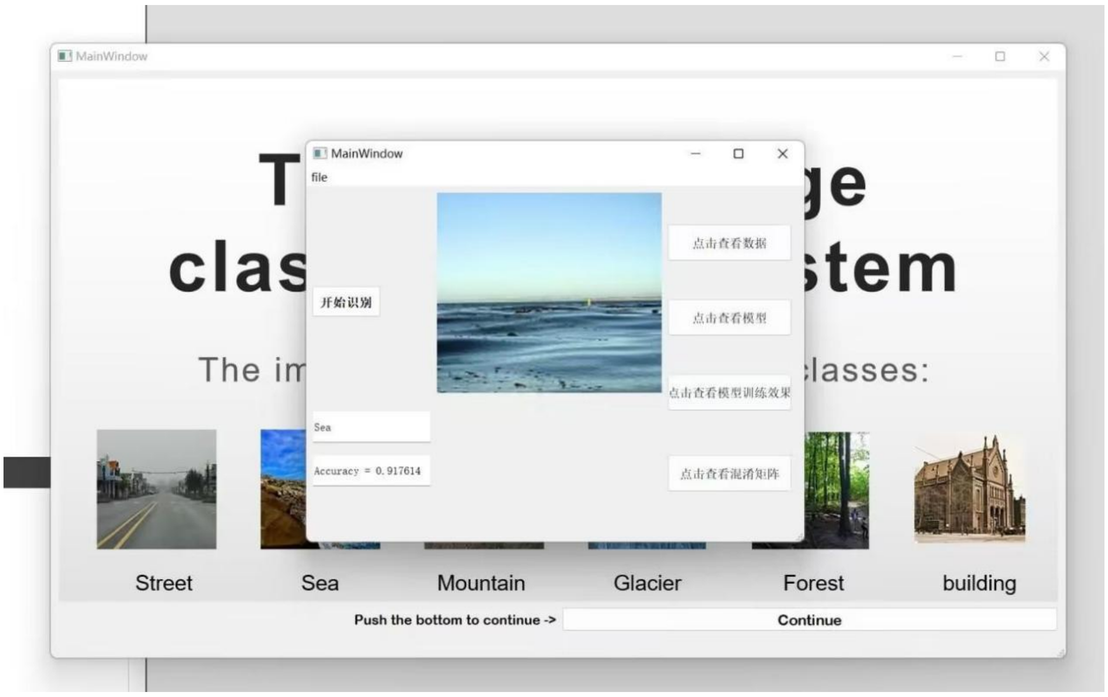


```
building
Accuracy = 0.941144
```

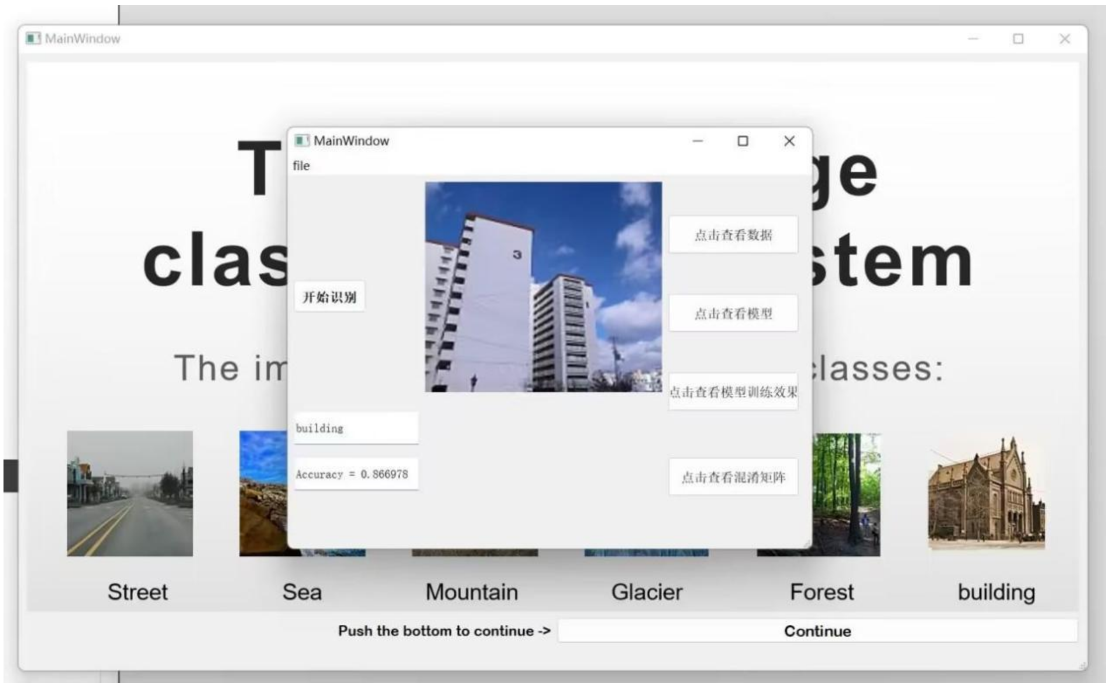


```
Forest
Accuracy = 1
```

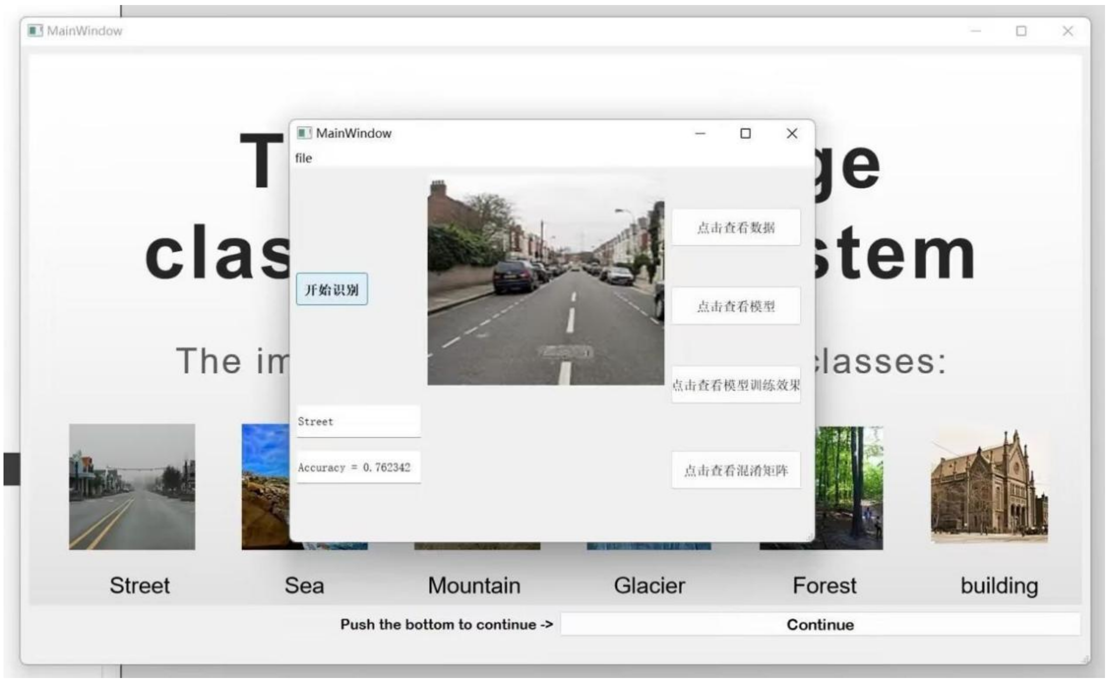
4.2.2 Image with arbitrary size



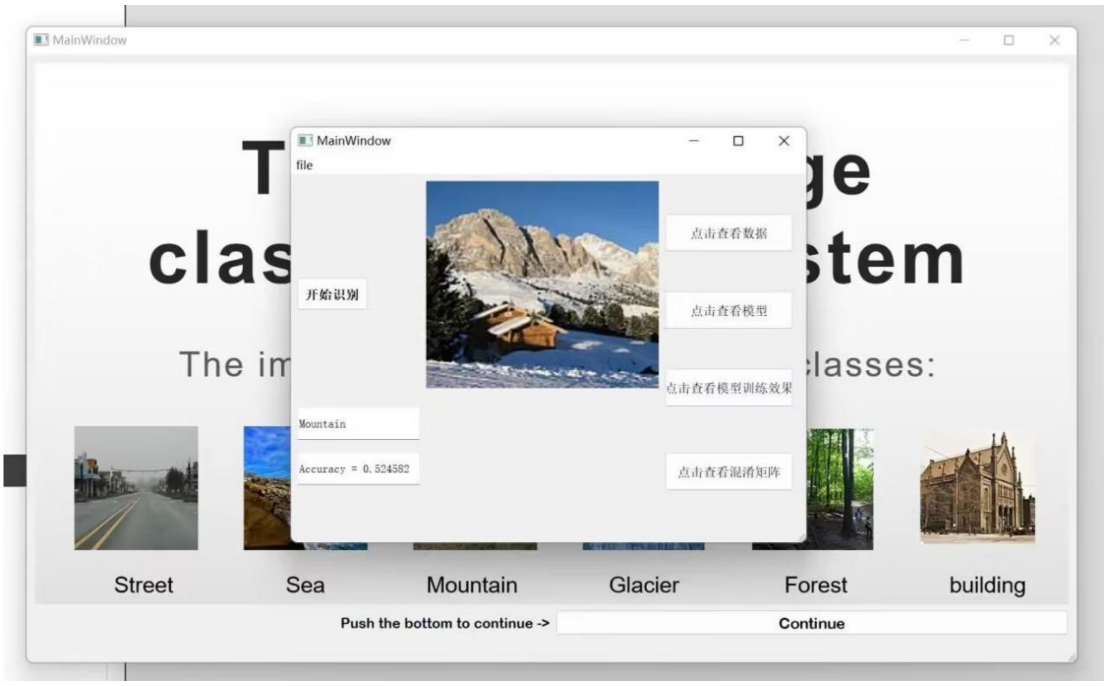
```
Sea
Accuracy = 0.917614
```



```
building
Accuracy = 0.866978
```



```
Street
Accuracy = 0.762342
```



It can be observed that the recognition performance of images with arbitrary sizes is slightly worse. This might be due to:

1. The model trained on 150x150 images is more suitable for recognizing square images.

2. Some data is lost during the size conversion of images with arbitrary sizes.

3. The structure of images with arbitrary sizes is more complex, compared to the simple structures of images in the test set (e.g., only houses, glaciers, etc.).

5. Conclusion

Through the construction of a CNN (Convolutional Neural Network), this course design essentially completes the deep learning of the Intel Image Classification database and outputs effective parameters, achieving an accuracy of 80%. It can convert any image into a 150x150 pixel value and classify it into one of the categories: buildings, streets, forests, glaciers, mountains, and oceans, and output the recognition accuracy. Furthermore (more content will be added later), we can conclude that the basic content of this course design has met the expected requirements.

Additionally, in the process of further improving the program, we will consider the following goals:

i) Multi-label learning: From the confusion matrix, we can see that the program is prone to confusing Glacier with Mountain and Street with Building. It's not difficult to deduce that this is because in some training and test examples, glaciers appear on mountains, and there are sometimes buildings on mountains; roads often have buildings, and the surroundings of buildings may also have streets. Multi-label learning can simultaneously recognize multiple labels, making it more practical in production and life.

ii) Using larger models: Employ larger models like ResNet50, VGGNet, or lightweight models like MobileNet, EfficientNet to achieve better fitting results. Regarding the use of ResNet50 and data augmentation, our team's code can be found at https://www.kaggle.com/code/charliestarstar/intel-image-resnet2 (it is best to use GPU or TPU for implementation).

iii) Creating a more beautiful UI: This primarily involves making the window size adaptive, designing the background, and creating more eye-catching buttons and labels.

6. Acknowledgments

We would like to conclude this final report by expressing our sincere gratitude to Professor Xing for his invaluable guidance and assistance throughout the programming process. Without his patient instruction, we would have encountered many more difficulties in our endeavors. Lastly, we would like to thank each member of our group: Wang Xinyue, Yao Yingfang, and Huang Tingwei. The successful completion of this program is a testament to the collective efforts of our team.

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. Appendix

Wang Xinyue: Neural network and parameter extraction, cooperation on the connection between Qt and the neural network, PPT, report
Yao Yingfang: Cooperation on the connection between Qt and the neural network, report
Huang Tingwei: Classification, Qt interface window, PPT, report