Multivariate Analysis Kaggle Competition

name: Dongwen Ou / student ID: 112652901

Exploratory Data Analysis (EDA)

> To start with, we need to read the image and label data into my Python environment.

In [1]:
```
import os
import pandas as pd
from PIL import Image
import numpy as np
from tqdm.notebook import tqdm
```

In [2]:
```
# 文件路径
train_img_path = '/Users/dongwenou/Downloads/ma_2024_1/train'
test_img_path = '/Users/dongwenou/Downloads/ma_2024_1/test'
train_csv_path = '/Users/dongwenou/Downloads/ma_2024_1/train.csv'
test_csv_path = '/Users/dongwenou/Downloads/ma_2024_1/test.csv'

# 读取CSV文件
train_df = pd.read_csv(train_csv_path)
test_df = pd.read_csv(test_csv_path)

print(train_df)
print(test_df)
```

```
        file_path  label
0      胎/胎_39.png      0
1       胎/胎_4.png      0
2       胎/胎_1.png      0
3      胎/胎_48.png      0
4      胎/胎_44.png      0
...           ...    ...
29732   攪/攪_0.png    999
29733  攪/攪_21.png    999
29734   攪/攪_6.png    999
29735  攪/攪_32.png    999
29736  攪/攪_40.png    999

[29737 rows x 2 columns]
            file_path
0          test/0.png
1          test/1.png
2          test/2.png
3          test/3.png
4          test/4.png
...               ...
22562  test/22562.png
22563  test/22563.png
22564  test/22564.png
22565  test/22565.png
22566  test/22566.png

[22567 rows x 1 columns]
```

For the below code chunk, I transferred my raw data into a list so that it can be easier for my later analysis.

In [39]:
```
# 加载训练集图片和标签
def load_train_images(train_df, train_img_path):
    images = []
    labels = []
    for i, row in train_df.iterrows():
        full_img_name = row['file_path']
        label = row['label']
        img_path = os.path.join(train_img_path, full_img_name)  # 构
        img = Image.open(img_path).convert('L')
        img_array = np.array(img)
        images.append(img_array)
        labels.append(label)
    return np.array(images), np.array(labels)

def load_test_images(test_df, test_img_path):
    images = []
    img_names = []
    for i, row in test_df.iterrows(): # iterrows() 是 pandas 的一个方
        img_name = row['file_path']  # 从当前行（row）中获取名为 file_p
        new_img_name = img_name.split('/')[-1]
        img_path = os.path.join(test_img_path, new_img_name)
        img = Image.open(img_path).convert('L')  # 转换为灰度模式（L）
        img_array = np.array(img)
        images.append(img_array)
        img_names.append(img_name)
    return np.array(images), img_names



# 读取训练集和测试集，是根据csv一行一行的读取的，所以训练集是一个字一个字往后排
train_images, train_labels = load_train_images(train_df, train_img_
test_images, test_img_names = load_test_images(test_df, test_img_pa
```

Now we can print them out to check their dimensions and oth
er information

In [15]:
```
print(train_images) #一张图是一个二维数组，白色数字大，黑色越小
print(train_labels)

print(f'Train Images: {train_images.shape}, Train Labels: {train_la
print(f'Test Images: {test_images.shape}')
```

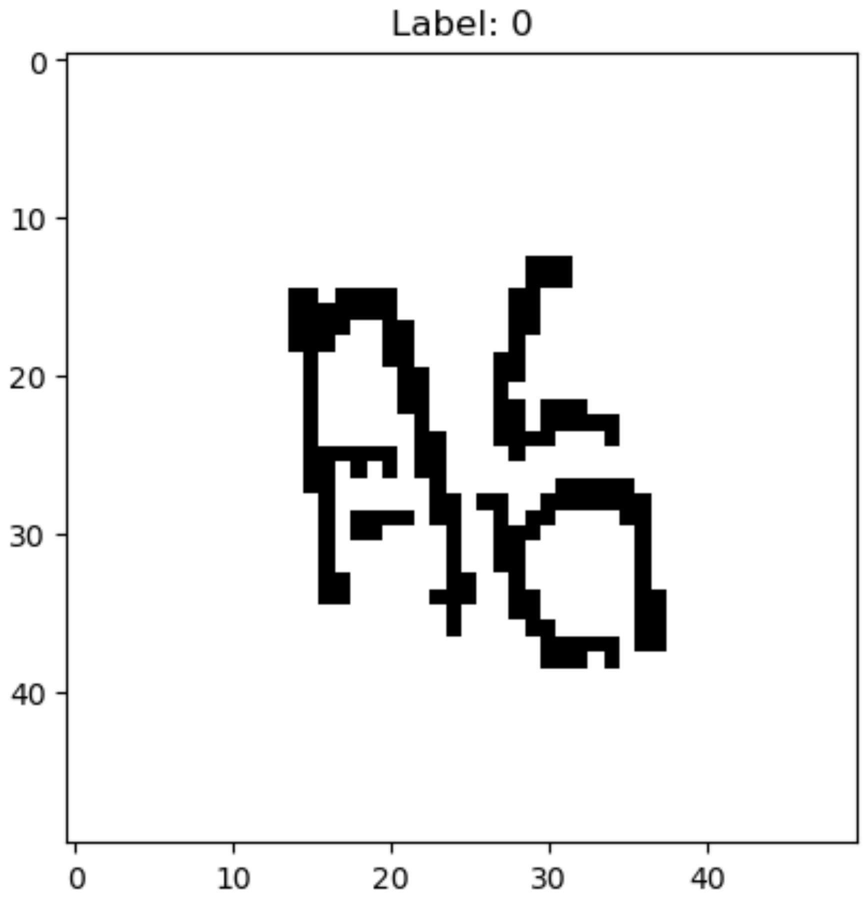
```
[[[255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  ...
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]]

 [[255 255 255 ... 255 255 255]
```

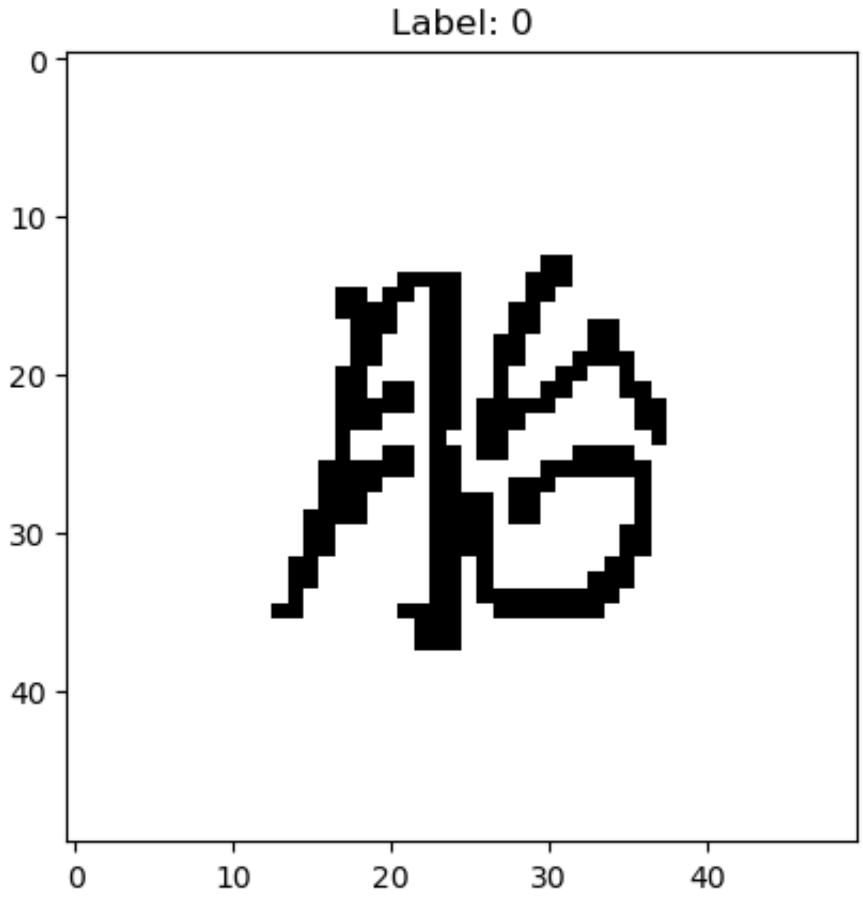
```
 [[255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  ...
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]]

 [[255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  ...
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]]

 ...

 [[255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  ...
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]]

 [[255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  ...
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]]

 [[255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  ...
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]]]
[  0   0   0 ... 999 999 999]
Train Images: (29737, 50, 50), Train Labels: (29737,)
Test Images: (22567, 50, 50)
```

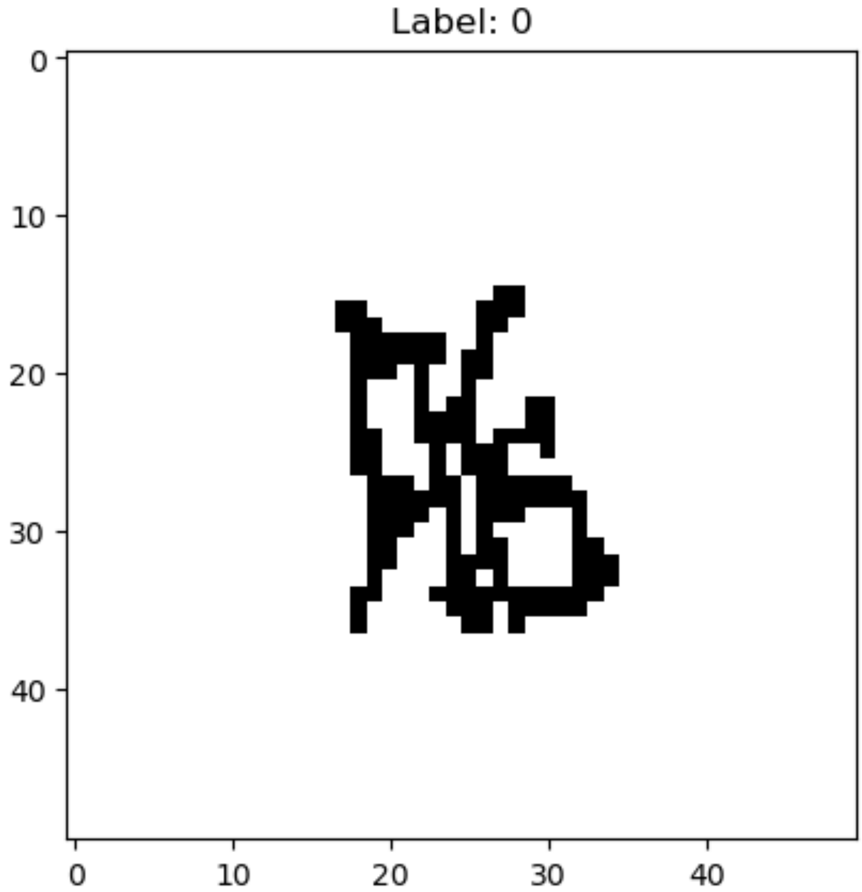
Now let's draw several images as an example.

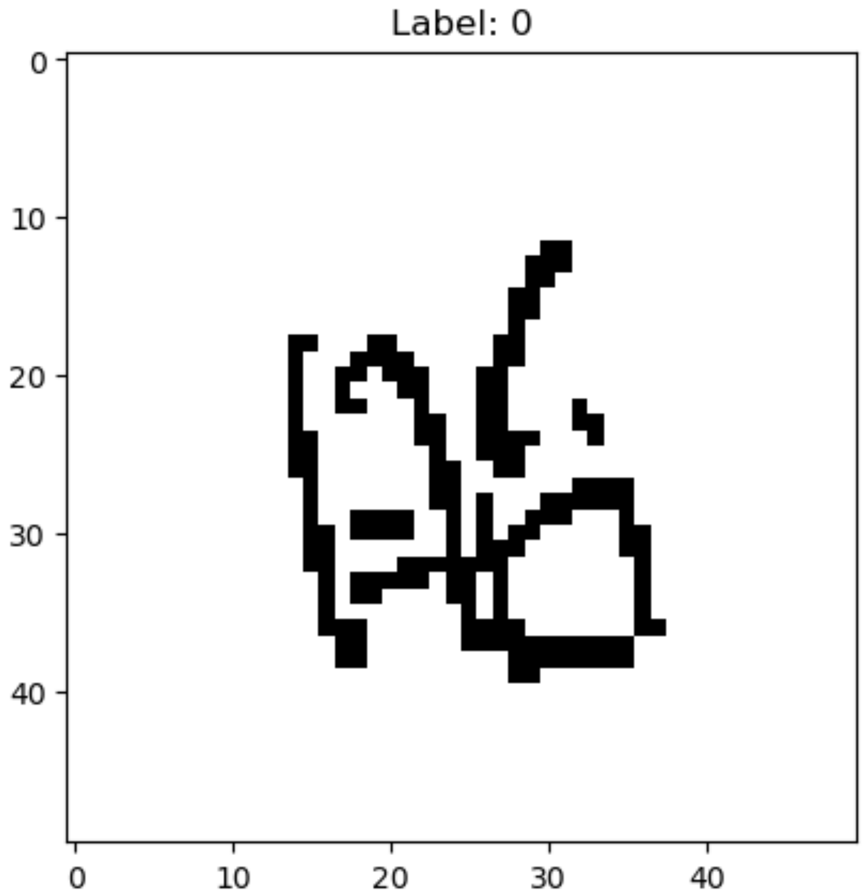
In [16]:
```
import matplotlib.pyplot as plt
# 显示前几个图像
for i in range(5):
    plt.imshow(train_images[i], cmap='gray')
```

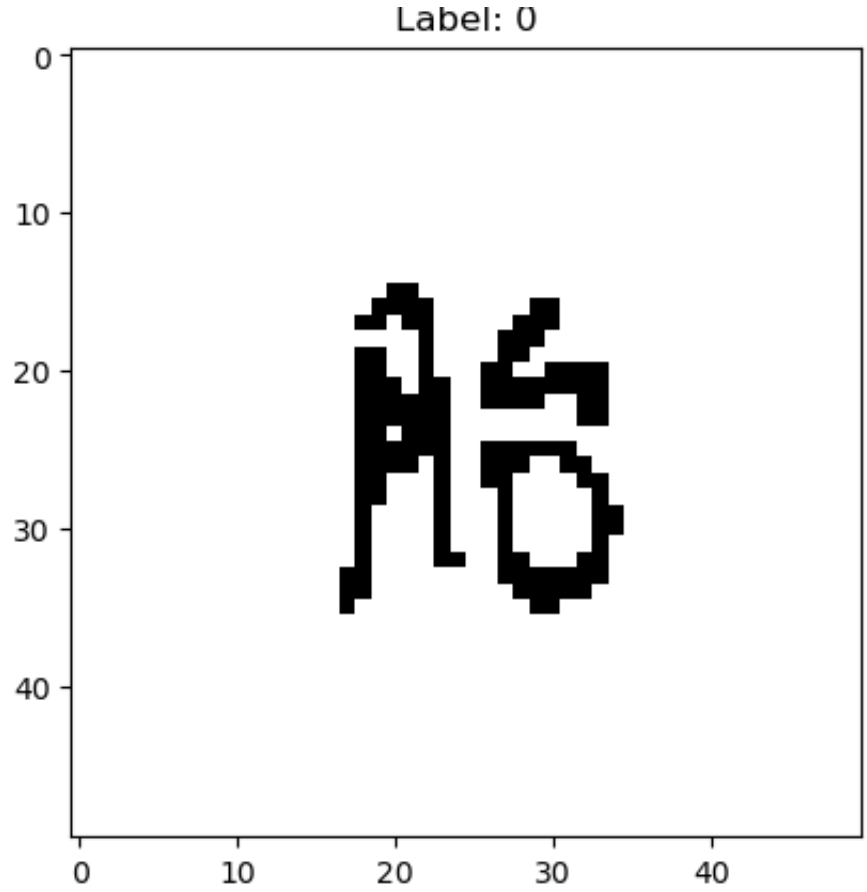
```
    plt.title(f'Label: {train_labels[i]}')
    plt.show()
```





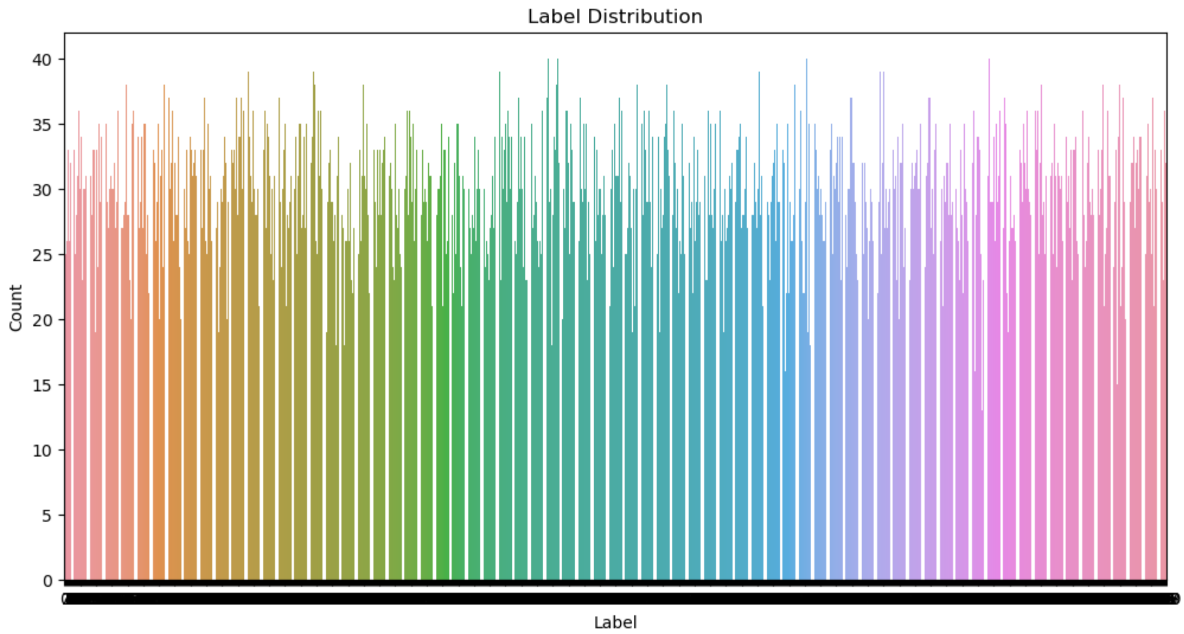

Label: 0
0
10
20
30
40
0
10
20
30
40


Label: 0
0
10
20
30
40
0
10
20
30
40



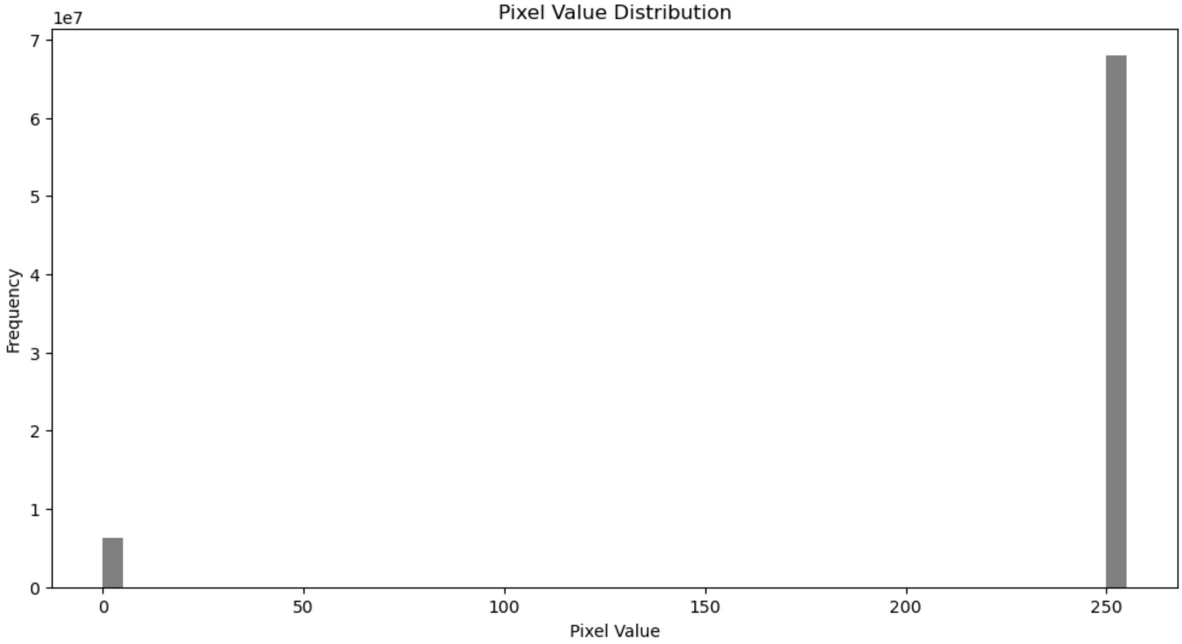
Then let's check the distributions of the training labels, see whether they're uniformly (evenly) distributed.

In [17]:
```
import seaborn as sns
# 将标签转换为 DataFrame 以便使用 Seaborn 进行可视化
labels_df = pd.DataFrame(train_labels, columns=['label'])

# 标签分布情况
plt.figure(figsize=(12, 6))
sns.countplot(x='label', data=labels_df)
plt.title('Label Distribution')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
```



Then let's check the pixels and see their statistical information, for which I found that they're not continuously distributed, but more like some binarized image pixels, which means they only have 0 and 255 as pixel values.

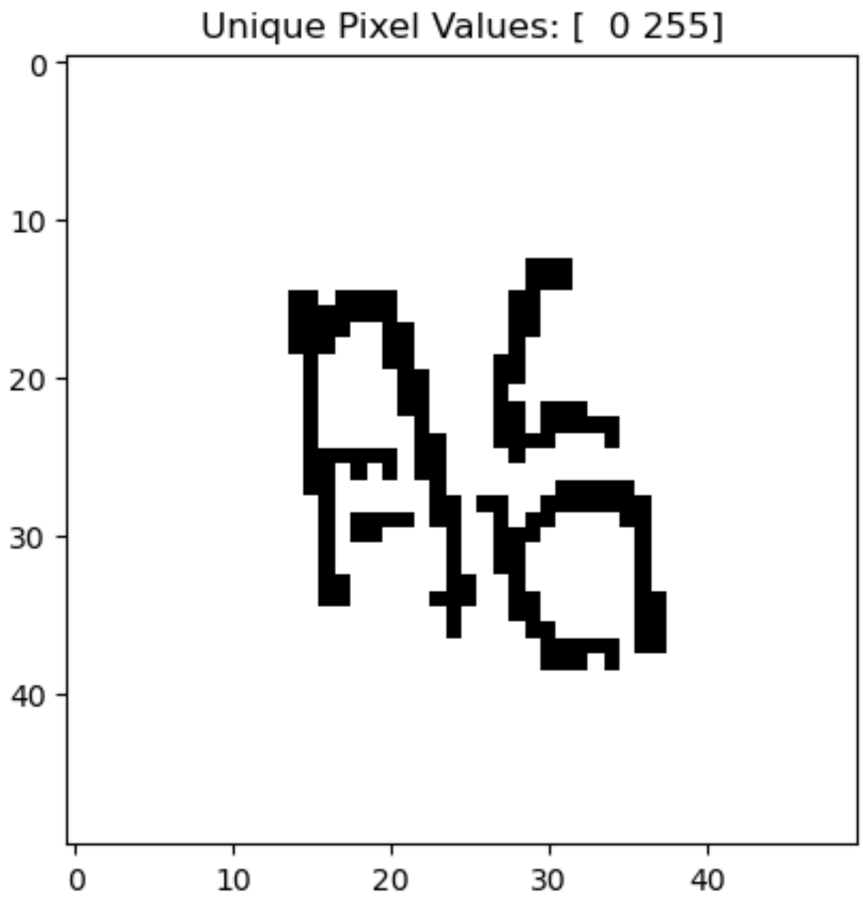
In [18]:
```
# 将图像数据展开为一维，以便查看像素值的统计信息
images_flattened = train_images.reshape(train_images.shape[0], -1)

# 查看像素值的统计信息
pixel_values = images_flattened.flatten()
plt.figure(figsize=(12, 6))
plt.hist(pixel_values, bins=50, color='gray')
plt.title('Pixel Value Distribution')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.show()
```



Above is just concluded from the chart, here I strictly pro
ved that the pixel valus are actually binary.

In [19]:
```
# 检查前几张图像的像素值唯一性
unique_pixel_values = [np.unique(image) for image in train_images[:
print(f"Unique pixel values in first 5 images: {unique_pixel_values

# 查看一张图像的像素值分布
image = train_images[0]
plt.imshow(image, cmap='gray')
plt.title(f'Unique Pixel Values: {np.unique(image)}')
plt.show()

# 将所有图像的像素值展平成一维数组
flattened_pixel_values = train_images.flatten()

# 计算像素值的唯一性
unique_pixel_values_in_dataset = np.unique(flattened_pixel_values)
print(f"Unique pixel values in dataset: {unique_pixel_values_in_dat
```

```
Unique pixel values in first 5 images: [array([  0, 255], dtype=ui
nt8), array([  0, 255], dtype=uint8), array([  0, 255], dtype=uint
8), array([  0, 255], dtype=uint8), array([  0, 255], dtype=uint
8)]
```



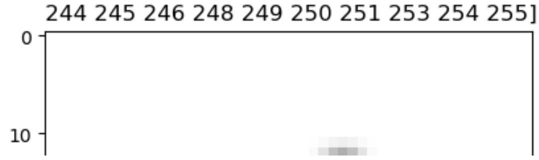

```
Unique pixel values in dataset: [  0 255]
```

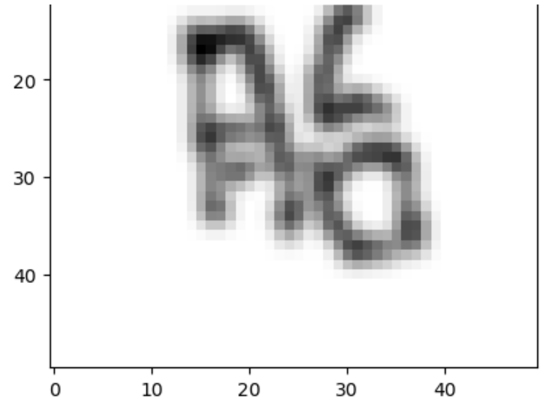
```
Then I apply Gaussian blur smoothing to the images because
this preprocessing might be instructive for improving the v
al_accuracy, but also maybe not, so I performed them both,
after which I'll choose the better performed one.
```

In [40]:

```python
#!pip install opencv-python-headless
import cv2

# 对图像进行高斯模糊平滑处理
smoothed_images = np.array([cv2.GaussianBlur(image, (5, 5), 0) for
# 对测试图像进行高斯模糊平滑处理
smoothed_test_images = np.array([cv2.GaussianBlur(image, (5, 5), 0)


# 查看平滑后的图像
smoothed_image = smoothed_images[0]
plt.imshow(smoothed_image, cmap='gray')
plt.title(f'Unique Pixel Values after Smoothing: {np.unique(smoothe
plt.show()

# 将平滑后的图像像素值展平成一维数组
flattened_smoothed_pixel_values = smoothed_images.flatten()

# 计算平滑后像素值的唯一性
unique_smoothed_pixel_values_in_dataset = np.unique(flattened_smoot
print(f"Unique pixel values after smoothing in dataset: {unique_smo


# 查看平滑后的测试图像
smoothed_test_image = smoothed_test_images[0]
plt.imshow(smoothed_test_image, cmap='gray')
plt.title(f'Unique Pixel Values after Smoothing (Test Image): {np.u
plt.show()

# 将平滑后的测试图像像素值展平成一维数组
flattened_smoothed_test_pixel_values = smoothed_test_images.flatten

# 计算平滑后测试图像像素值的唯一性
unique_smoothed_test_pixel_values_in_dataset = np.unique(flattened_
print(f"Unique pixel values after smoothing in test dataset: {uniqu
```

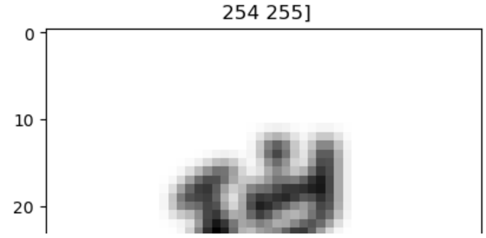
Unique Pixel Values after Smoothing: [33 44 48 49 63 67 74 76 77 78 79 80 85 87 88 89 90 91
92 93 94 95 96 97 99 100 101 102 104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119 120 121 123 124 125 126 127 128 129 130
131 133 134 135 138 139 140 141 144 145 146 147 148 149 150 151 152 153
154 155 156 157 158 159 160 161 163 164 165 166 168 169 170 171 173 174
177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194
195 196 197 198 199 200 201 204 205 208 209 210 211 213 214 215 216 218
219 220 223 224 225 228 229 230 231 232 233 235 236 238 239 240 241 243
244 245 246 248 249 250 251 253 254 255]

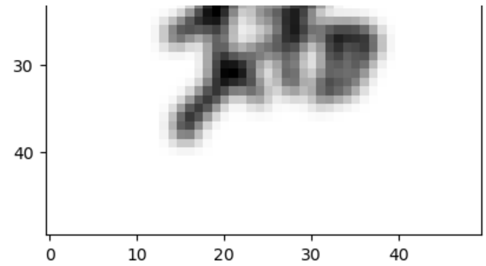




```
Unique pixel values after smoothing in dataset: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255]
```

Unique Pixel Values after Smoothing (Test Image): [20 21 24 37 41 42 44 45 46 49 50 51 56 58 60 61 62 64
65 66 67 68 70 71 72 73 74 75 76 77 78 80 81 83 84 85
86 87 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104
105 106 107 108 109 110 111 113 115 116 117 118 119 120 121 122 124 125
126 127 128 129 130 133 134 135 136 137 139 140 141 142 143 144 145 146
147 148 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165
166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191 192 194 195 196 199 200 201 202 204 205
206 207 208 209 210 213 214 215 216 217 218 219 220 221 223 224 225 226
229 230 233 234 235 236 237 238 239 240 241 244 245 246 249 250 251 253
254 255]






```
Unique pixel values after smoothing in test dataset: [  0   1   2
3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33
34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69
70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87
88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 1
06 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 1
24 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 1
42 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 1
60 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 1
78 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 1
96 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 2
14 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 2
32 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 2
50 251
 252 253 254 255]
```

In [ ]:

Now we'll move on to model selection. At the very beginning, I only used the models in classic Machine Learning such as SVM, KNN, Random Forest and Logistic Regression. Unfortunately, it performed really poor, with an accuracy rate lower than 50%.

Consequently, I chose CNN as it's almost the best classifier for images, and let's see the details as following,

In [7]:
```
#用CNN訓練模型
#pip install tensorflow
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Fl
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
```

Method one

```
Here I'll use the unchanged images to be my training data.
```

data preparation:

```
Firstly, I standardized the values of each pixels to be wit
hin the interval [0, 1].

Secondly, while I was trying to split the data into trainin
g and validation sets, I used "train_test_split" to randoml
y select. However, I came to realize that some Chinese char
acters might have too many or too few as testing images, th
en I split the data manually as follows, where I pick 20% a
s testing data.

At last, I used one-hot encoding to process the testing dat
a in order for the use of CNN model.
```

In [8]:
```
# 数据预处理
train_images = train_images / 255.0
test_images = test_images / 255.0

# 将标签和图像数据组合
data = list(zip(train_images, train_labels))

# 按标签分组
data_by_label = {}
for img, label in data:
    if label not in data_by_label:
        data_by_label[label] = []
    data_by_label[label].append(img)

# 初始化训练集和验证集
train_images_split = []
train_labels_split = []
val_images_split = []
val_labels_split = []

# 从每个标签中抽取20%的样本作为验证集
for label, images in data_by_label.items():
    val_size = int(0.2 * len(images))
    val_indices = np.random.choice(len(images), val_size, replace=F

    val_images_split.extend([images[i] for i in val_indices])
    val_labels_split.extend([label] * val_size)

    train_indices = [i for i in range(len(images)) if i not in val_
    train_images_split.extend([images[i] for i in train_indices])
    train_labels_split.extend([label] * len(train_indices))

# 转换为numpy数组
X_train = np.array(train_images_split)
y_train = np.array(train_labels_split)
X_val = np.array(val_images_split)
y_val = np.array(val_labels_split)

# 将标签转换为one-hot编码
num_classes = len(np.unique(y_train))
y_train_hot = to_categorical(y_train, num_classes)
y_val_hot = to_categorical(y_val, num_classes)
```

Model description:

    Below is the design of each layer of my CNN model in detai
    l. Specifically, the network starts with an input layer tha
    t accepts training images. The first layer is a convolution
    al layer with 64 filters of size 3x3, followed by a Leaky R
    eLU activation function with an alpha of 0.15 and a dropout
    layer with a rate of 0.2, which helps prevent overfitting.

    Next, there is another convolutional layer with 64 filters
    of size 3x3, followed by the same Leaky ReLU activation and
    a max-pooling layer to reduce the spatial dimensions. This
    is followed by another dropout layer with a rate of 0.2. Th
    e next two are similar which just differ in the number of f
    ilters.

    Finally, the output layer is a dense layer with 1000 units
    and a softmax activation function, which provides a probabi
    lity distribution over 1000 possible classes for classifica
    tion. This architecture combines convolutional layers for f
    eature extraction with fully connected layers for classific
    ation, incorporating dropout layers to mitigate overfittin
    g.

In [22]:

```
#最好好好！！！！！

from tensorflow.keras.layers import LeakyReLU, Activation
# 检查数据分布
print(f'X_train shape: {X_train.shape}')
print(f'y_train_hot shape: {y_train_hot.shape}')
print(f'X_val shape: {X_val.shape}')
print(f'y_val_hot shape: {y_val_hot.shape}')

model = Sequential([
    Input(shape=(50, 50, 1)),

    Conv2D(64, kernel_size=(3, 3)),
    LeakyReLU(alpha=0.15),
#    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.2),

    Conv2D(64, kernel_size=(3, 3)),
    LeakyReLU(alpha=0.15),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.2),

    Conv2D(128, kernel_size=(3, 3)),
```

```
    LeakyReLU(alpha=0.15),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.3),

    Conv2D(256, kernel_size=(3, 3)),
    LeakyReLU(alpha=0.15),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.4),


    Flatten(),

    Dense(256),
    LeakyReLU(alpha=0.15),
    Dropout(0.5),

    Dense(128),
    LeakyReLU(alpha=0.15),
    Dropout(0.5),

    Dense(1000, activation='softmax')
])

# 编译模型
model.compile(optimizer='adam', loss='categorical_crossentropy', me
```

```
X_train shape: (24188, 50, 50)
y_train_hot shape: (24188, 1000)
X_val shape: (5549, 50, 50)
y_val_hot shape: (5549, 1000)

/opt/anaconda3/lib/python3.11/site-packages/keras/src/layers/activ
ations/leaky_relu.py:41: UserWarning: Argument `alpha` is deprecat
ed. Use `negative_slope` instead.
  warnings.warn(
```

Results and discussions:

```
Below is the execution part of my model, I used early stopp
ing and model checking to help get the best model. However,
in this way, the model building process is quite time-consu
ming, but can reach an accuracy rate of approximately 0.96
5.

With using the CNN model, I adjusted every parameters in or
der to maximize my val_accuracy. I used different activatio
n function: Relu, Leaky Relu, Swish. I tried different desi
gn of each layer. For instance, like the number of filters,
kernel size or whether to set a MaxPooling, and the dropout
rate for sure.

For the dense layer, I use activation of softmax in order t
o get the probability, as I need to label the images that w
ere not in the training dataset previously as '-1'.

As a result, I gradually searched for the best parameters.
I didn't use grid search as the running time could explode,
or different combination might need to be dealt with differ
ent standards. But finally, I guess this is the final accur
acy - 0.96. I found that the number of layers and filters c
annot be too many, or else the model could reach poor perfo
rmance and the running time would cost a few days, so the m
ain thought is to maximize the val_accuracy while keeping t
he model relatively simple.
```

In [10]:
```
mport tensorflow as tf
rom tensorflow.keras.models import Sequential
rom tensorflow.keras.layers import Conv2D, LeakyReLU, ReLU, MaxPool
rom tensorflow.keras.preprocessing.image import ImageDataGenerator
rom tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopp
无加强数据

rom tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoin

 定义 EarlyStopping 回调, 监控 val_accuracy, 如果在 5 个 epochs 内没有提高
arly_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=99,
    restore_best_weights=True,  # 恢复 val_accuracy 最高的模型权重
    verbose=1
```

```
# 定义 ModelCheckpoint 回调，在训练过程中保存 val_accuracy 最高的模型
model_checkpoint = ModelCheckpoint(
    filepath='best_model.keras',
    monitor='val_accuracy',
    save_best_only=True,
    verbose=1


lr_scheduler = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.8,
    patience=15,
    min_lr=0.00002,
    verbose=1


with tf.device('/GPU:0'):
    history = model.fit(
        X_train, y_train_hot,
        validation_data=(X_val, y_val_hot),
        epochs=1000,  # 可以设定一个较大的值，因为我们使用 EarlyStopping
        batch_size=128,
        callbacks=[early_stopping, model_checkpoint, lr_scheduler]
    )
```

```
oss: 0.1377 - val_accuracy: 0.9618 - val_loss: 0.1674 - learning_r
ate: 1.6777e-04
Epoch 352/1000
189/189 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step - accuracy: 0.9558 - lo
ss: 0.1406
Epoch 352: val_accuracy did not improve from 0.96396
189/189 ━━━━━━━━━━━━━━━━━━━━ 51s 268ms/step - accuracy: 0.9558 - l
oss: 0.1406 - val_accuracy: 0.9627 - val_loss: 0.1626 - learning_r
ate: 1.6777e-04
Epoch 353/1000
189/189 ━━━━━━━━━━━━━━━━━━━━ 0s 251ms/step - accuracy: 0.9559 - lo
ss: 0.1414
Epoch 353: val_accuracy did not improve from 0.96396

Epoch 353: ReduceLROnPlateau reducing learning rate to 0.000134217
73910522462.
189/189 ━━━━━━━━━━━━━━━━━━━━ 50s 266ms/step - accuracy: 0.9559 - l
oss: 0.1414 - val_accuracy: 0.9625 - val_loss: 0.1591 - learning_r
ate: 1.6777e-04
Epoch 354/1000
```

In [11]:

```
# 加载保存的最佳模型
best_model = tf.keras.models.load_model('best_model.keras')
```

Prediction

```
After I got the best model, I used it to predict the testin
g data. Here I set the threshold probability to be 0.5, whi
ch means like the posterior, if the prob of an image belong
ing to every label is all below 0, then we should conclude
that it may not ever appear in the training dataset.
```

In [12]:
```python
# 预测测试数据
test_predictions_prob = best_model.predict(test_images)
test_predictions = np.argmax(test_predictions_prob, axis=1)
print(test_predictions_prob)
# 设置概率阈值
probability_threshold = 0.5

# 处理训练集中不存在的类别（假设这些类别为-1）
final_predictions = []

for i, pred in enumerate(test_predictions):
    if max(test_predictions_prob[i]) < probability_threshold:
        final_predictions.append(-1)
    else:
        final_predictions.append(pred)

# 创建提交文件
submission_df = pd.DataFrame({
    'ID': [f'test/{i}.png' for i in range(len(test_images))],
    'label': final_predictions
})

submission_df.to_csv('submission.csv', index=False)
```

```
706/706 ━━━━━━━━━━━━━━━━━━━━ 13s 18ms/step
[[7.98501612e-16 7.80495799e-15 2.16344829e-18 ... 9.21917013e-17
  1.30131270e-15 7.83479885e-20]
 [5.28878358e-29 4.25258916e-23 2.09721814e-22 ... 4.93190198e-24
  9.18454177e-21 2.21466325e-23]
 [2.72781931e-16 2.81123156e-25 1.56595343e-23 ... 1.30153403e-22
  1.19277910e-30 2.54505398e-30]
 ...
 [8.76163147e-20 3.37177222e-18 6.38998964e-32 ... 2.60989327e-16
  1.38182991e-35 1.37894839e-29]
 [3.28640459e-24 3.21331698e-22 8.18115230e-20 ... 6.84624713e-15
  1.55358740e-35 2.18105449e-26]
 [2.37869019e-26 4.67157561e-25 1.21237285e-36 ... 3.00071952e-29
  1.68064747e-20 3.98870225e-23]]
```

Method two

For this case, we'll use the smoothed images instead of the binary ones as our training data.
Moreover, the change below is just in the first row of codes, and the model building training and testing are just the same.

In [41]:
```
# 数据预处理
train_images = smoothed_images / 255.0
test_images = smoothed_test_images / 255.0

# 将标签和图像数据组合
data = list(zip(train_images, train_labels))

# 按标签分组
data_by_label = {}
for img, label in data:
    if label not in data_by_label:
        data_by_label[label] = []
    data_by_label[label].append(img)

# 初始化训练集和验证集
train_images_split = []
train_labels_split = []
val_images_split = []
val_labels_split = []

# 从每个标签中抽取20%的样本作为验证集
for label, images in data_by_label.items():
    val_size = int(0.2 * len(images))
    val_indices = np.random.choice(len(images), val_size, replace=F

    val_images_split.extend([images[i] for i in val_indices])
    val_labels_split.extend([label] * val_size)

    train_indices = [i for i in range(len(images)) if i not in val_
    train_images_split.extend([images[i] for i in train_indices])
    train_labels_split.extend([label] * len(train_indices))

# 转换为numpy数组
X_train = np.array(train_images_split)
y_train = np.array(train_labels_split)
X_val = np.array(val_images_split)
y_val = np.array(val_labels_split)

# 将标签转换为one-hot编码
num_classes = len(np.unique(y_train))
y_train_hot = to_categorical(y_train, num_classes)
y_val_hot = to_categorical(y_val, num_classes)
```

In [29]:
```
model = Sequential([
    Input(shape=(50, 50, 1)),

    Conv2D(64, kernel_size=(3, 3)),
    LeakyReLU(alpha=0.15),
    #MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.2),

    Conv2D(64, kernel_size=(3, 3)),
    LeakyReLU(alpha=0.15),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.2),

    Conv2D(128, kernel_size=(3, 3)),
    LeakyReLU(alpha=0.15),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.3),

    Conv2D(256, kernel_size=(3, 3)),
    LeakyReLU(alpha=0.15),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.4),


    Flatten(),

    Dense(256),
    LeakyReLU(alpha=0.15),
    Dropout(0.5),

    Dense(128),
    LeakyReLU(alpha=0.15),
    Dropout(0.5),

    Dense(1000, activation='softmax')
])

# 编译模型
model.compile(optimizer='adam', loss='categorical_crossentropy', me
```

In [30]:
```
# 定义 EarlyStopping 回调，监控 val_accuracy，如果在 5 个 epochs 内没有提
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=99,
    restore_best_weights=True,  # 恢复 val_accuracy 最高的模型权重
    verbose=1
)

# 定义 ModelCheckpoint 回调，在训练过程中保存 val_accuracy 最高的模型
model_checkpoint = ModelCheckpoint(
    filepath='best_model.keras',
    monitor='val_accuracy',
    save_best_only=True,
    verbose=1
)

lr_scheduler = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.8,
    patience=15,
    min_lr=0.00002,
    verbose=1
)

with tf.device('/GPU:0'):
    history = model.fit(
        X_train, y_train_hot,
        validation_data=(X_val, y_val_hot),
        epochs=1000,  # 可以设定一个较大的值，因为我们使用 EarlyStopping
        batch_size=128,
        callbacks=[early_stopping, model_checkpoint, lr_scheduler]
    )
```

```
oss: 0.2148 - val_accuracy: 0.9504 - val_loss: 0.2056 - learning_r
ate: 1.3422e-04
Epoch 273/1000
189/189 ━━━━━━━━━━━━━━━━━━━━ 0s 248ms/step - accuracy: 0.9385 - lo
ss: 0.2017
Epoch 273: val_accuracy did not improve from 0.95152
189/189 ━━━━━━━━━━━━━━━━━━━━ 50s 262ms/step - accuracy: 0.9385 - l
oss: 0.2017 - val_accuracy: 0.9512 - val_loss: 0.1951 - learning_r
ate: 1.3422e-04
Epoch 274/1000
189/189 ━━━━━━━━━━━━━━━━━━━━ 0s 248ms/step - accuracy: 0.9390 - lo
ss: 0.1911
Epoch 274: val_accuracy did not improve from 0.95152
189/189 ━━━━━━━━━━━━━━━━━━━━ 50s 263ms/step - accuracy: 0.9390 - l
oss: 0.1912 - val_accuracy: 0.9411 - val_loss: 0.2532 - learning_r
ate: 1.3422e-04
Epoch 275/1000
189/189 ━━━━━━━━━━━━━━━━━━━━ 0s 248ms/step - accuracy: 0.9337 - lo
ss: 0.2081
Epoch 275: val_accuracy did not improve from 0.95152
```

In [34]:
```
# 加载保存的最佳模型
best_model1 = tf.keras.models.load_model('best_model.keras')
```

In [42]:
```
# 预测测试数据
test_predictions_prob = best_model1.predict(test_images)
test_predictions = np.argmax(test_predictions_prob, axis=1)
print(test_predictions_prob)
# 设置概率阈值
probability_threshold = 0.5

# 处理训练集中不存在的类别（假设这些类别为-1）
final_predictions = []

for i, pred in enumerate(test_predictions):
    if max(test_predictions_prob[i]) < probability_threshold:
        final_predictions.append(-1)
    else:
        final_predictions.append(pred)

# 创建提交文件
submission_df = pd.DataFrame({
    'ID': [f'test/{i}.png' for i in range(len(test_images))],
    'label': final_predictions
})

submission_df.to_csv('submission.csv', index=False)
```

```
706/706 ━━━━━━━━━━━━━━━━━━━━ 12s 18ms/step
[[1.52996120e-15 6.50749325e-15 5.62864577e-17 ... 2.08487551e-16
  6.94868095e-15 4.28985088e-15]
 [9.09326971e-23 1.06186716e-21 1.53908862e-15 ... 4.06107811e-20
  3.37892891e-17 7.32282906e-16]
 [2.76000475e-17 5.24397945e-27 1.64753490e-23 ... 1.16953185e-20
  1.92306266e-33 2.50667862e-29]
 ...
 [9.16857261e-15 4.10256173e-14 5.27580157e-20 ... 1.43655785e-13
  6.46292963e-30 2.09412950e-21]
 [9.19531708e-26 1.11274582e-29 1.99167452e-15 ... 2.27301000e-16
  2.51406191e-33 2.26487488e-23]
 [1.47798440e-25 1.09032183e-24 2.10164945e-25 ... 7.96566933e-21
  3.43268188e-18 9.25298047e-18]]
```

From all of the above analysis, maybe the second method should have a better behavior, but I realized this point too late and due to the time limitation I failed to discover the most suitable model for the continuous case.

As a result, Method one outperformed the other one, that model will be chosen as the classifier.

In [ ]:

```

```