Handwritten Chinese Character Recognition

[1] *Chun-Wei Tseng* (曾俊為), [1,*]*Chiou-Shann Fuh* (傅楸善)

[1]Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan
[*]E-mail: champion8599@gmail.com fuh@csie.ntu.edu.tw

ABSTRACT

Since the convolutional neural network (CNN) has been widely used in the field of computer vision, it has been introduced to the handwritten Chinese character recognition. In this paper, we simply use traditional CNN model with some common techniques like data augmentation, max pooling, dropout and oversampling to do recognition, we finally achieve 93.69% accuracy on over 10,000 validation image data and 88.18% accuracy on 1,600 testing image data.

Keywords: handwritten Chinese character recognition; Convolutional Neural Network; data augmentation; oversampling

1. RELATED WORKS

Optical Character Recognition (OCR) is the most common technique in pattern recognition, it has been widely used as a form of data entry from printed paper data records, such as passport documents, bank statements, business cards.

Neural Network is commonly used at machine learning; we can use it to solve problems and optimize result via mathematical statistics and learning methods.

2. DATA PRE-PROCESSING

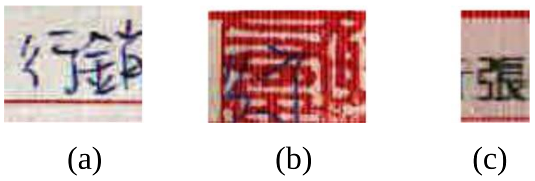


Fig. 1. Noisy images.

There are many problems with the images in Figure 1. For example, the images classified as "行". For Figure 1(a), there are two words in an image. It is not allowed in the competition, for Figure 1(b), there are many noises like the red part that can easily affect training results. For Figure 1(c), it is even not the correct word.

There are many red noises in most of the noised images. First, we try using the distribution of the pixel values in the image, and it successfully removed most of the noisy images in Figure 1(b).

Then we try using cosine similarity between different images in a category to get rid of the wrong images, such as Figure 1(a). Unfortunately, the results are not reliable, so we turn to clean the image data manually.

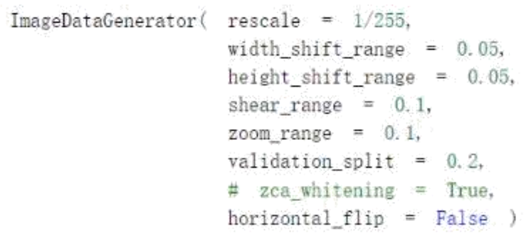
Thus, the remaining data can be clean.

3. METHODS

In deep learning, a large amount of data is needed to ensure that overfitting does not occur during training. Therefore, the following methods are generally used to solve the overfitting problem caused by insufficient information.

1. regularization,
2. dropout techniques,
3. data augmentation.

We decide to do image data augmentation in Figure 2, and there is a kit from Keras that can easily meet our requirements, ImageDataGenerator.



```
ImageDataGenerator(  rescale  =  1/255,
                     width_shift_range  =  0.05,
                     height_shift_range  =  0.05,
                     shear_range  =  0.1,
                     zoom_range  =  0.1,
                     validation_split  =  0.2,
                     #  zca_whitening  =  True,
                     horizontal_flip  =  False  )
```

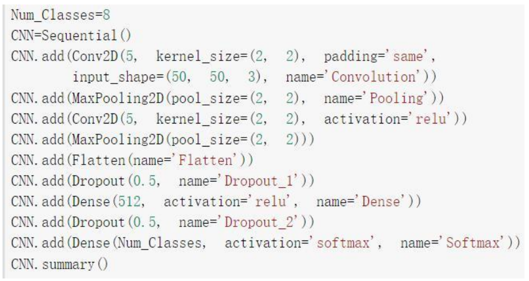
Fig. 2. Image Data Generator code segment.

It provides parameters that can be chosen to modify. First, we rescale pixel values to 0~1. It is conducive to the convergence of the model, then we set width_shift_range and height_shift_range to 0.05. Both of them control the amount of translation of the image, and we set shear_range and zoom_range to 0.1, they can also control the content of the image. At last, we set validation_split to 0.2 for the validation dataset.

After image data augmentation, our next step is to build a CNN model. We set two convolution layers combined with max pooling layers. The role of the pooling layer is to reduce noises and computing resources, and then we flatten the feature maps into vectors.

Also, we use ReLU as our activation function, and we use softmax function in the output layer for multiple classification. Adding dropout is another way to prevent overfitting problem that has been mentioned above.

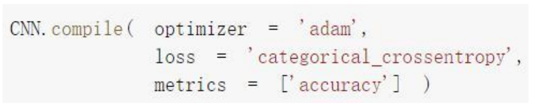
Our initial code segment in Figure 3.



```
Num_Classes=8
CNN=Sequential()
CNN.add(Conv2D(5,  kernel_size=(2,  2),  padding='same',
        input_shape=(50,  50,  3),  name='Convolution'))
CNN.add(MaxPooling2D(pool_size=(2,  2),  name='Pooling'))
CNN.add(Conv2D(5,  kernel_size=(2,  2),  activation='relu'))
CNN.add(MaxPooling2D(pool_size=(2,  2)))
CNN.add(Flatten(name='Flatten'))
CNN.add(Dropout(0.5,  name='Dropout_1'))
CNN.add(Dense(512,  activation='relu',  name='Dense'))
CNN.add(Dropout(0.5,  name='Dropout_2'))
CNN.add(Dense(Num_Classes,  activation='softmax',  name='Softmax'))
CNN.summary()
```

Fig. 3. Initial CNN model code segment.

Besides, we use Adam as our optimizer of learning rate, and we choose categorical cross-entropy as our loss in CNN model.



```
CNN.compile(  optimizer  =  'adam',
              loss  =  'categorical_crossentropy',
              metrics  =  ['accuracy']  )
```

Fig. 4. CNN model compile code segment.

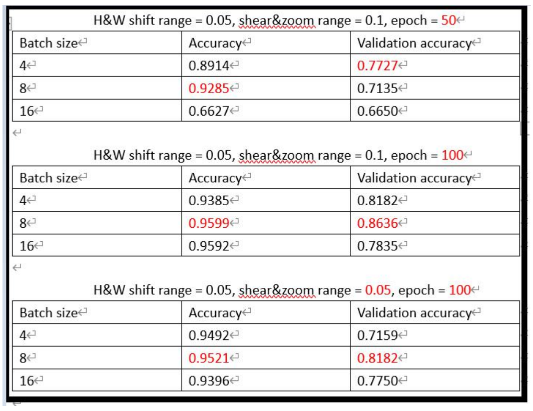
4. EXPERIMENTS

First, we choose 4 simple Chinese characters for experiment. It performs well so we turn to choose 4 complex Chinese characters, and it performs well also.

Then we combine 4 simple characters and 4 complex characters as our training set. At the beginning, both training accuracy and validation accuracy is about 70%, and it performs badly at testing data.

Therefore, we modify some of the parameters of the model, such as batch size, epoch, and the amount of the data augmentation.

After trial and error in Figure 5, we achieve 95% accuracy in Figure 6 and performs nearly perfectly on testing data in Figure 7.



H&W shift range = 0.05, shear&zoom range = 0.1, epoch = 50

Batch size	Accuracy	Validation accuracy
4	0.8914	0.7727
8	0.9285	0.7135
16	0.6627	0.6650

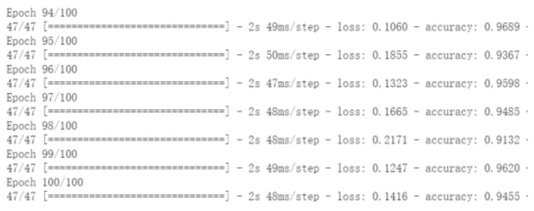
H&W shift range = 0.05, shear&zoom range = 0.1, epoch = 100

Batch size	Accuracy	Validation accuracy
4	0.9385	0.8182
8	0.9599	0.8636
16	0.9592	0.7835

H&W shift range = 0.05, shear&zoom range = 0.05, epoch = 100

Batch size	Accuracy	Validation accuracy
4	0.9492	0.7159
8	0.9521	0.8182
16	0.9396	0.7750

Fig. 5. Accuracy on training data.



```
Epoch 94/100
47/47 [==============================] - 2s 49ms/step - loss: 0.1060 - accuracy: 0.9689 ·
Epoch 95/100
47/47 [==============================] - 2s 50ms/step - loss: 0.1855 - accuracy: 0.9367 ·
Epoch 96/100
47/47 [==============================] - 2s 47ms/step - loss: 0.1323 - accuracy: 0.9598 ·
Epoch 97/100
47/47 [==============================] - 2s 48ms/step - loss: 0.1665 - accuracy: 0.9485 ·
Epoch 98/100
47/47 [==============================] - 2s 48ms/step - loss: 0.2171 - accuracy: 0.9132 ·
Epoch 99/100
47/47 [==============================] - 2s 49ms/step - loss: 0.1247 - accuracy: 0.9620 ·
Epoch 100/100
47/47 [==============================] - 2s 48ms/step - loss: 0.1416 - accuracy: 0.9455 ·
```

Fig. 6. The result of CNN training.

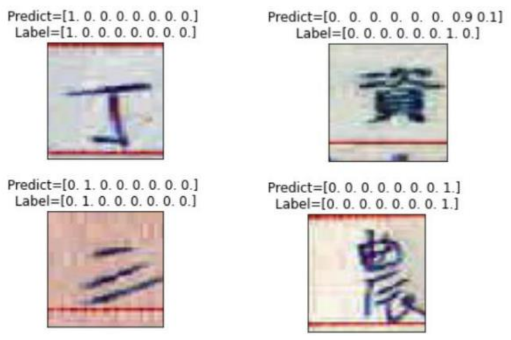

Predict=[1. 0. 0. 0. 0. 0. 0. 0.]
Label=[1. 0. 0. 0. 0. 0. 0. 0.]
Predict=[0. 0. 0. 0. 0. 0. 0.9 0.1]
Label=[0. 0. 0. 0. 0. 0. 1. 0.]
Predict=[0. 1. 0. 0. 0. 0. 0. 0.]
Label=[0. 1. 0. 0. 0. 0. 0. 0.]
Predict=[0. 0. 0. 0. 0. 0. 0. 1.]
Label=[0. 0. 0. 0. 0. 0. 0. 1.]

Fig. 7. The result on testing data.

And then we try bigger data size, we found that the accuracy is still good when the amount of the training categories is about 200, but the accuracy gets worse and worse when the number of categories become bigger.

Thus, we try to modify the parameter of the model, we change the set of ImageDataGenerator, shear_range is set to 0.2 and zoom range is set to 0.2, and we add rotation to the training images, we set the rotation range to 15 degrees.

After these steps above, we found that these settings are the best to our propose to make the training data more complex but still maintain the difference between different characters.

Then we found that the size of the images is not fixed and may also affect the result, so we try many ways to solve the problem.

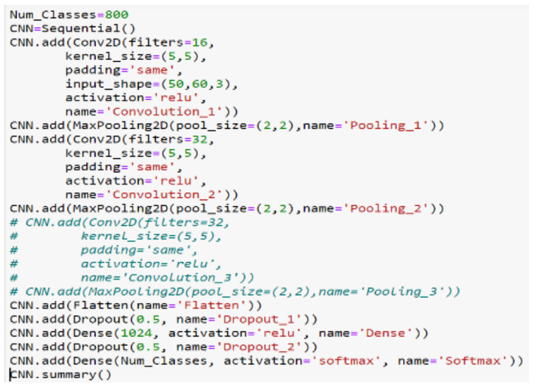
First, we try fully-connected convolution layers, we found that although the problem is solved but the training result is not good enough.

We turn to try setting only one image in a batch, but unfortunately the training time increase a lot, but the accuracy does not prove it is worth.

Then we try padding different images to fixed-size, after many experiments, we finally set the image size to (50, 60) pixels.

Finally, we modify the structure and the parameters of the CNN model, which is the most important to the whole training process.

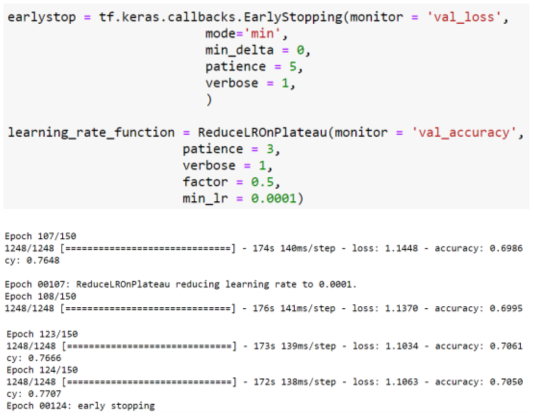
Due to the number of categories of the characters given by the host of the competition, 800. Deeper neural network, bigger kernel size or more filters may be needed. After many experiments on the whole training dataset, we finally decide our CNN model in Figure 8.



```
Num_Classes=800
CNN=Sequential()
CNN.add(Conv2D(filters=16,
        kernel_size=(5,5),
        padding='same',
        input_shape=(50,60,3),
        activation='relu',
        name='Convolution_1'))
CNN.add(MaxPooling2D(pool_size=(2,2),name='Pooling_1'))
CNN.add(Conv2D(filters=32,
        kernel_size=(5,5),
        padding='same',
        activation='relu',
        name='Convolution_2'))
CNN.add(MaxPooling2D(pool_size=(2,2),name='Pooling_2'))
# CNN.add(Conv2D(filters=32,
#         kernel_size=(5,5),
#         padding='same',
#         activation='relu',
#         name='Convolution_3'))
# CNN.add(MaxPooling2D(pool_size=(2,2),name='Pooling_3'))
CNN.add(Flatten(name='Flatten'))
CNN.add(Dropout(0.5, name='Dropout_1'))
CNN.add(Dense(1024, activation='relu', name='Dense'))
CNN.add(Dropout(0.5, name='Dropout_2'))
CNN.add(Dense(Num_Classes, activation='softmax', name='Softmax'))
CNN.summary()
```

Fig. 8. CNN model code segment.

We also add Early Stopping and reduce Learning Rate in the training process. It makes training time less and the convergence of validation accuracy fast and accurate in Figure 9.



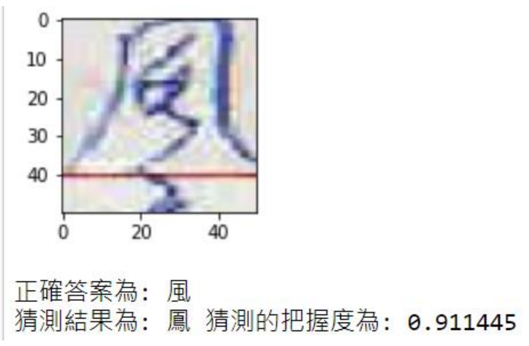
```
earlystop = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss',
                             mode='min',
                             min_delta = 0,
                             patience = 5,
                             verbose = 1,
                             )

learning_rate_function = ReduceLROnPlateau(monitor = 'val_accuracy',
                          patience = 3,
                          verbose = 1,
                          factor = 0.5,
                          min_lr = 0.0001)
```

```
Epoch 107/150
1248/1248 [==============================] - 174s 140ms/step - loss: 1.1448 - accuracy: 0.6986
cy: 0.7648

Epoch 00107: ReduceLROnPlateau reducing learning rate to 0.0001.
Epoch 108/150
1248/1248 [==============================] - 176s 141ms/step - loss: 1.1370 - accuracy: 0.6995

Epoch 123/150
1248/1248 [==============================] - 173s 139ms/step - loss: 1.1034 - accuracy: 0.7061
cy: 0.7666
Epoch 124/150
1248/1248 [==============================] - 172s 138ms/step - loss: 1.1063 - accuracy: 0.7050
cy: 0.7707
Epoch 00124: early stopping
```

Fig. 9. EarlyStopping and ReduceLR code segment and results.

On May 29, 2021, we achieve about 90% accuracy on validation data and 81.2% accuracy on 1,600 testing data and rank the 25th among more than 400 contestants on test match.

We wish to solve the missed guessing problem in Figure 10 which is a common problem in character recognition and get better place in the final competition.



正確答案為: 凨
猜測結果為: 鳳 猜測的把握度為: 0.911445

Fig. 10. Wrong guess on testing data.

Later, we try more on the training data and modify the parameters of the CNN model.

First, we change the color of the images from RGB to grayscale in Figure 11. Since the change of the input dimension of the convolution layer, we have to modify and get another pair of parameters, it is another big work here.

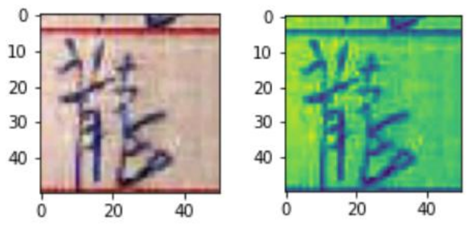


Fig. 11. RGB image and grayscale image.

As the result, we cannot get better accuracy with grayscale images, so we turn to add more images to training data, because most of the images in the competition look like left of Figure 11, so we put blue characters and yellow in the background with some noises in our handcrafted images to simulate original images.

Therefore, we create and label more images to each category, about 7,000 images among all 800 categories.

On June 13, 2021, we achieve about 92% accuracy on validation data and 84% accuracy on 1,600 testing data.

There are still few days left before the final competition, we might try adding new categories that do not belong to the original data to improve our result on handwritten character recognition.

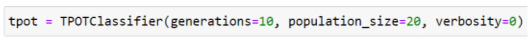
After the competition, we find that we failed on classifying the "isnull" category, the character that is not belong to the given 800 categories, we think it happens because we only use degree of certainty as our threshold, so we decide to solve the problem.

We add the images given by the competition official that do not belong to the given 800 categories to the "isnull" category, and then we also add some of the most common characters that are not belong to the given 800 categories to the "isnull" category, we confirm it will perform better in the competition.

At last, we find that the imbalance of the training data may also causes serious problem, so we try under-sampling and oversampling to the training data. For under-sampling, we delete some images from categories that have bigger quantities; for oversampling, we copy existing images from categories that have smaller quantities, the goal of both methods is to balance the amount of the training data.

After many experiments on both samplings, although the result of under-sampling is slightly better than oversampling, but we choose oversampling as our method to maintain the variety of the training data.

Finishing all these steps above, we use Tpot, a machine learning kit from scikit-learn to fine-tune our parameters in Figure 12.

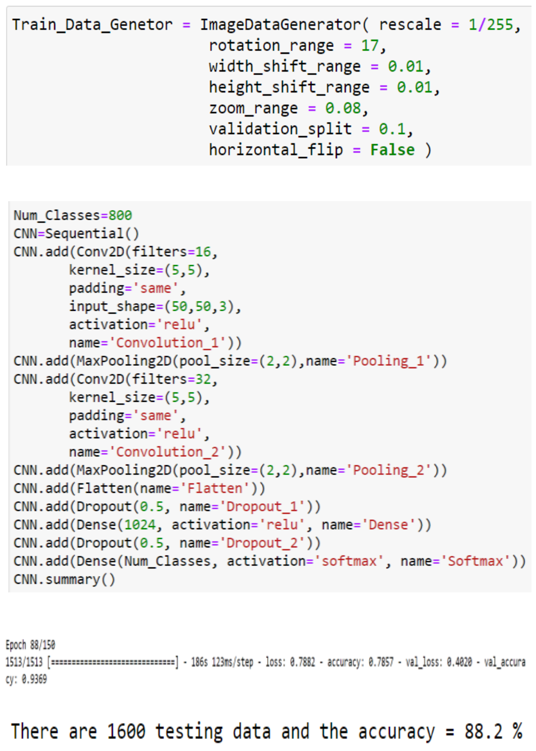


```
tpot = TPOTClassifier(generations=10, population_size=20, verbosity=0)
```

Fig. 12. Tpot classifier.

With our final version of model, we achieve 93.69% accuracy on over 10,000 validation data and 88.18% accuracy on 1,600 testing data.

There is our final version of our CNN model and results of training and testing.



```
Train_Data_Genetor = ImageDataGenerator( rescale = 1/255,
                         rotation_range = 17,
                         width_shift_range = 0.01,
                         height_shift_range = 0.01,
                         zoom_range = 0.08,
                         validation_split = 0.1,
                         horizontal_flip = False )
```

```
Num_Classes=800
CNN=Sequential()
CNN.add(Conv2D(filters=16,
       kernel_size=(5,5),
       padding='same',
       input_shape=(50,50,3),
       activation='relu',
       name='Convolution_1'))
CNN.add(MaxPooling2D(pool_size=(2,2),name='Pooling_1'))
CNN.add(Conv2D(filters=32,
       kernel_size=(5,5),
       padding='same',
       activation='relu',
       name='Convolution_2'))
CNN.add(MaxPooling2D(pool_size=(2,2),name='Pooling_2'))
CNN.add(Flatten(name='Flatten'))
CNN.add(Dropout(0.5, name='Dropout_1'))
CNN.add(Dense(1024, activation='relu', name='Dense'))
CNN.add(Dropout(0.5, name='Dropout_2'))
CNN.add(Dense(Num_Classes, activation='softmax', name='Softmax'))
CNN.summary()
```

```
Epoch 88/150
1513/1513 [==============================] - 186s 123ms/step - loss: 0.7882 - accuracy: 0.7857 - val_loss: 0.4020 - val_accura
cy: 0.9369
```

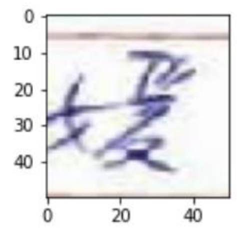
```
There are 1600 testing data and the accuracy = 88.2 %
```

Fig. 13. Final CNN model and results.

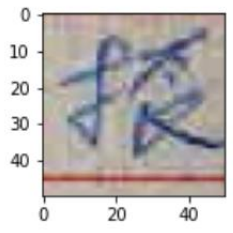
5. RESULTS

We divide our recognition results into four categories.

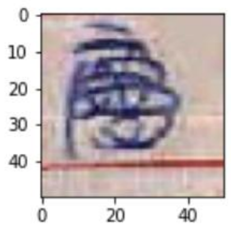
1. Correct and confident recognition



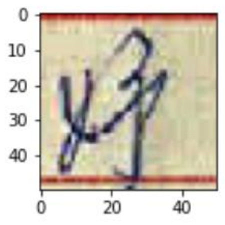
```
正確答案為: 媛
猜測結果為: 媛 猜測的把握度為: 0.9764346
```



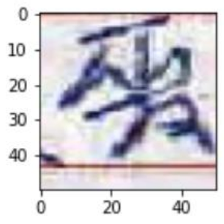
```
正確答案為: 振
猜測結果為: 振 猜測的把握度為: 0.96826035
```



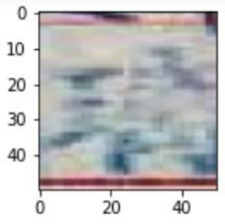
```
正確答案為: 慶
猜測結果為: 慶 猜測的把握度為: 0.9488474
```



```
正確答案為: 妤
猜測結果為: 妤 猜測的把握度為: 0.9319867
```

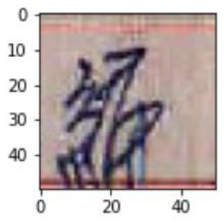


```
正確答案為: 晉
猜測結果為: 晉 猜測的把握度為: 0.8977465
```

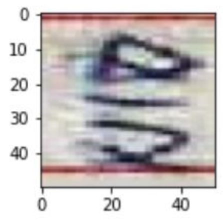


```
正確答案為: 璃
猜測結果為: 璃 猜測的把握度為: 0.81981486
```

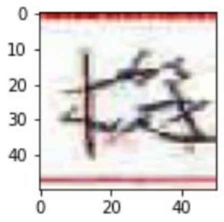
2. Correct recognition without high confidence



```
正確答案為: 綸
猜測結果為: 綸 猜測的把握度為: 0.36167005
```

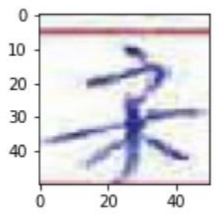


```
正確答案為: 旻
猜測結果為: 旻 猜測的把握度為: 0.36122143
```

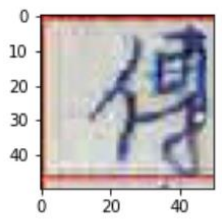


```
正確答案為: 機
猜測結果為: 機 猜測的把握度為: 0.33443454
```

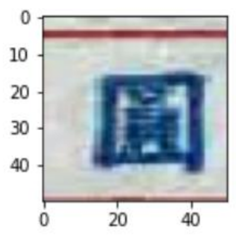
3. Incorrect but explainable recognition



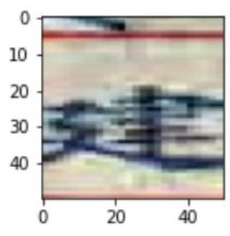
```
正確答案為: 宋
猜測結果為: 柔 猜測的把握度為: 0.8983956
```



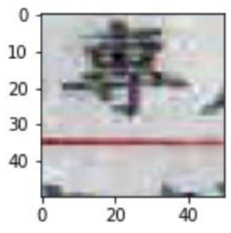
```
正確答案為: 傅
猜測結果為: 得 猜測的把握度為: 0.61900514
```



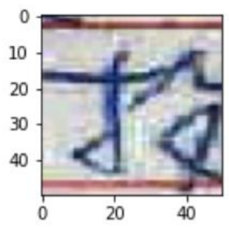
```
正確答案為: 圓
猜測結果為: 園 猜測的把握度為: 0.5612169
```



```
正確答案為: 建
猜測結果為: 連 猜測的把握度為: 0.499543
```

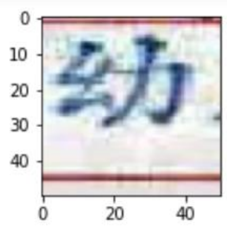


```
正確答案為: 專
猜測結果為: 壽 猜測的把握度為: 0.3344415
```

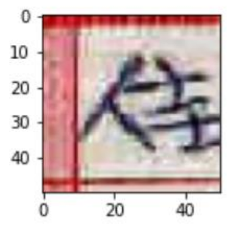


```
正確答案為: 投
猜測結果為: 技 猜測的把握度為: 0.3013046
```

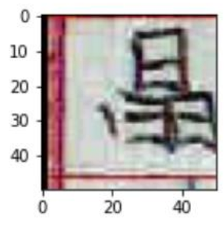
4. Incorrect and unexplainable recognition



```
正確答案為: 幼
猜測結果為: 訊 猜測的把握度為: 0.73011345
```



```
正確答案為: 佳
猜測結果為: 緯 猜測的把握度為: 0.39532343
```



```
正確答案為: 晶
猜測結果為: 萬 猜測的把握度為: 0.29597503
```

6. CONCLUSION

We simply use CNN as our model for handwritten Chinese character recognition, for data preprocessing, we clean the data manually and we do data augmentation combined with image creation for various image data, we also do oversampling to balance the amount of the data; for training, we add two convolution layers with max pooling layers in neural network and also do batch normalization and dropout for preventing overfitting, in addition, early stopping and declining learning rate are added in training to improve our performance and reduce the convergence time of CNN training.

In the future, we hope to try more parameters to fine-tune techniques or more interesting models on handwritten Chinese character recognition like RCNN, ResNetV2 and so on.

REFERENCES

[1] Ji Gan, Weiqiang Wang, Ke Lu, Compressing the CNN architecture for in-air handwritten Chinese character recognition, Pattern Recognition Letters (2020).

[2] Zhouyao Zhong, Lianwen Jin, Zecheng Xie, High performance offline handwritten Chinese character recognition using GoogLeNet and directional feature maps, IEEE (2015).

[3] Meijun He, Shuye Zhang, Huiyun Mao, Lianwen Jin, Recognition confidence analysis of handwritten Chinese character with CNN, IEEE (2015).

[4] Qingqing Wang, Yue Lu, Similar Handwritten Chinese Character Recognition Using Hierarchical CNN Model, IEEE (2017).

[5]https://keras.io/api/layers/regularization_layers/dropout/

[6]https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

[7]https://www.tensorflow.org/tutorials/images/cnn

[8] https://keras.io/api/layers/pooling_layers/

[9] https://keras.io/api/callbacks/early_stopping/

[10]https://keras.io/api/callbacks/reduce_lr_on_plateau/

[11]https://towardsdatascience.com/oversampling-and-undersampling-5e2bbaf56dcf

[12] http://epistasislab.github.io/tpot/api/

[13] https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[14] https://keras.io/api/applications/resnet/