Training classifiers for recognition of adults and children

ZEHUA GUO (21336130), ZHIPENG JIA (20327934), YAHONG ZHU (21333370)

1 Introduction

This project aims to implement a binary classifier for distinguishing adults and children. To achieve this target, we have used a model of CNN and Logistic Regression as our baseline. And we chose this topic for two reasons. First of all, we considered that dealing with images is much more interesting than dealing with the texts because images are more visually diverse and attractive than texts. Therefore, we choose a topic related to image processing. Moreover, in reality, there are huge amounts of scenes that need to distinguish between adults and children. For example, some places such as bars and KTVs should not allow children to enter. Another example is that some places provide free services only for children, while adults have to pay before entering. If there is a classifier that can help people complete these classification tasks, it could save lots of labor.

The input of our classifier is a group of images including children and adults and then we use a CNN model to output a predicted label representing the classification result of the image. Similarly, we input the same images into a Logistic Regression model and compare the results with those obtained by the CNN model. According to our conjecture, CNN will show a better performance over the Logistic Regression.

2 Dataset and Features

Our dataset consists of 18,000 grayscale 48×48 images, 50% for children and 50% for adults. And in each group, we use 7,200 images (80%) for training and the rest 1,800 images (20%) for testing. We collected and preprocessed these images according to the following steps.

2.1 The whole flow of data preprocessing

First, we wrote the code to grab the images from the Internet. To grab images, we take advantage of the several python libraries including the **urllib**, **re**, and **os**. The urllib library can help us to call the [**urlopen]** function, which allows us to open the URL and returns the results as a file-like object. And then we compiled a regular expression pattern [**r'https://[^\s]*?\.jpg'**] by the function [**compile**] of **re**, which provides regular expression matching operations. In that case, we could find all the links that match this pattern on the page. After getting the linklist, we create a local folder by library **os** and download the images through each JPG link by the function [**urlretrieve**] provided by **urllib**, which can retrieve a URL into a temporary location on disk. Until now, we have finally done the work of grabbing images from the Internet. In addition, many large-scale image websites have perfect security measures to avoid being captured by such methods. For this reason, we had to find a website that does not have these security limitations and has plenty of image resources. And fortunately, we have found one ideal website called **shutterstock** and grabbed about 70,000 raw images in total.

https://www.shutterstock.com/search/child+face?page=1

https://www.shutterstock.com/search/adult+face?page=1

2.2 Example URL Link for grabbing Images

Second, we cleaned up the data, including removing some duplicate images and the images without faces. Since there were so many duplicate images in the raw images and thus it was difficult to get rid of them manually. Therefore, we solved this problem by using python programming. The main method of the procedure is to judge if the images are duplicate or not and save the unique ones only. The discriminant for judgment is shown as below:

$$\frac{1}{n}\sqrt{\sum_i^n \quad (h_1(i) - h_2(i))^2} \leq threshold$$

The h in the formula above represents each pixel value in the source image, we got it through the **histogram** function of the Python Imaging Library (**PIL**). After several simple tests, we found that when threshold=1, duplicate images could be identified exactly. In other words, we judged if the two images are the same by comparing whether the Mean Square Error of the two images' pixels is less than or equal to 1. Although we succeeded in getting rid of the duplicate images, there were still many inefficient images. And after this step, only less than 50% (about 30,000) of our images are left.

Third, we look through the rest of the images and get rid of most useless images manually and then use the **Cascade Classifier**, a function provided by OpenCV, which is most commonly used in image processing for object detection and tracking, primarily facial detection and recognition[1] to help us to recognize and capture faces. Then, we resized each image to 48×48 pixels. The example of processing results is shown as Figure 2.3 below. At the end of this step, we get about 6,000 faces of adults and 6,000 faces of children.

(A) Adult (Right: Input Left: Output)

(B) Child(Right: Input Left: Output)

Figure 2.3 Result of Step3

Finally, we check the faces once again and delete the noisy images. This time, there are only 4,500 faces left in each group. In this way, images for training neural networks seem to be insufficient, so we double the images set by flipping the image horizontally like Figure 2.4 and we have got 9,000 adults' faces and 9,000 children's faces eventually. In addition, training CNN models is very time-consuming, so we considered making the size of our images as small as possible. With this in mind, we converted all the RGB images to yCbCr images and then retained the luminance channel only in order to change the RGB image into grayscale images.

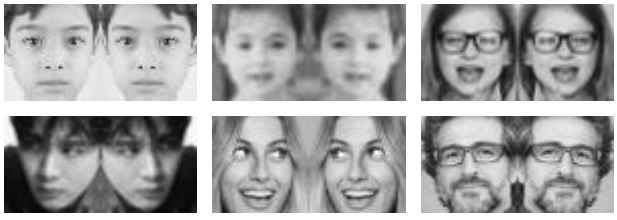


Figure 2.4 Result of Step4

Through the 4 steps mentioned above, we have obtained our training and testing dataset. Then we put them in four folders named Adults_test, Adults_train, Children_test, Children_train. Before training, we read them separately from folders and labeled the child face with 0, labeled the adult face with 1, and then made our x_train(images set), y_train(label set) for training, x_test(images set) for predicting, and y_test(label) for evaluating the performance of the model.

[1] https://en.wikipedia.org/wiki/Cascading_classifiers

3 Methods

I. Convolutional Neural Network (CNN)

CNNs are a type of non-linear model that is most used for processing images, for example, image recognition, segmentation, and classification. It contains convolutional layers, pooling layers, and fully connected layers. The input is a tensor with a certain shape, then, put it into the convolutional layer to get the feature map. There are pooling layers that follow the convolutional layers to reduce the dimensions of the features. Usually, there are two types of pooling layers, max-pooling which calculates the maximum feature of the feature map, and average-pooling which calculates the average of n×n features on the map, where n represents the kernel size. The fully connected layer has the same structure as the multi-layer perceptron (MLP) which connects every node from its input layer to every node of the next layer. The input of fully connected layers is a flattened tensor, and the output is the probability of every class. CNNs can extract features automatically from images by the convolutional layer and learn these features through training, whereas traditional algorithms extract features manually. This is one of the advantages that makes CNN is suitable for processing a large number of images with high accuracy. We are going to train the CNN model with relu activation function for convolutional layers, softmax for fully connected layers, adam for optimizing, and categorical cross-entropy for the loss function, which will be explained below.

The problem we are going to solve is a binary image classification for distinguishing children and adults. Therefore, CNNs are the best choice for solving these kinds of problems.

II. PCA & Logistic Regression (Baseline)

Principal Component Analysis (**PCA**) is a common and effective method to reduce the dimensionality of the input data. For images, it calculates eigenvalues and corresponding eigenvectors of the covariance matrix of input images whose shape is n×n where n represents the number of features. All eigenvalues will then be sorted in descending order to obtain eigenvectors with the biggest k eigenvalues to form a matrix P. Applying P following $Y = PX$ results in a matrix Y containing vectorized images with reduced dimensionality. PCA is useful when there are a great number of features in each training sample.

Logistic Regression is a common linear model for classification problems. The basic idea of Logistic Regression is to find a decision boundary that can linearly separate the whole dataset into two parts with each part being classified into one category. The expression of Logistic Regression model can be written as $h_\theta(x) = \theta^T x$, where n represents the number of features in the input vector x. The cost function for Logistic Regression is $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \log\left(1 + e^{-y^{(i)}\theta^T x^{(i)}}\right)$. The training process then is simply to minimize $J(\theta)$ using optimization algorithms. When using scikit learn to train Logistic Regression models, the Broyden-Fletcher-Goldfarb-Shanno algorithm is used by default, which has great robustness against an unscaled dataset and is suitable for relatively small dataset[2].

4 Experiments/Results/Discussion

Our whole experimental process is based on the following configuration:

- *11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz / NVIDIA GeForce RTX 3050 Ti Laptop GPU*
- *TensorFlow: ver. 2.3.0*

I. Convolutional Neural Network (CNN)

This CNN contains 20,346 parameters in total, which has 6 convolutional layers with relu activation function, 3 max-pooling layers, and one fully connected layer. The dropout layer randomly drops 50% of neurons to reduce parameters. The fully connected layer uses the softmax activation function to output 2 predictions. For

[2] https://scikit-learn.org/stable/modules/linear_model.html#id29

evaluation, we use categorical cross-entropy loss function which is used for true labels are one-hot encoded. It's described as $Loss = -\sum_{i=1}^{output\ size} y_i\ log\hat{y}_i$. Then, adam optimizer is applied to find a global minimum.

Table 4.1 Architecture of ConvNet

Layer	Input Shape	Kernel size, channel	Output Shape, channel
Conv Layer1	48×48×1	3×3×3, 8	48×48×1, 8
Conv Layer2	48×48×1, 8	3×3×3, 8	48×48×1, 8
Max_pooling	48×48×1, 8	2×2, padding = 'same'	24×24×1, 8
Conv Layer3	24×24×1, 8	3×3×3, 16	24×24×1, 16
Conv Layer4	24×24×1, 16	3×3×3, 16	24×24×1, 16
Max_pooling	24×24×1, 16	2×2, padding = 'same'	12×12×1, 16
Conv Layer5	12×12×1, 16	3×3×3, 32	12×12×1, 32
Conv Layer6	12×12×1, 32	3×3×3, 32	12×12×1, 32
Max_pooling	12×12×1, 32	2×2, padding = 'same'	6×6×1, 32
Dropout	6×6×1, 32	50%	6×6×1, 32
Flatten	6×6×1, 32	-	1152
FC Layer	1152	-	2
Total parameters: 20,346			

For training the model, we use 7200 adult images and 7200 children's images, and for testing, we use 1800 adult images and 1800 children's images. We trained the ConvNet in 30 epoch and the batch size is 64. For

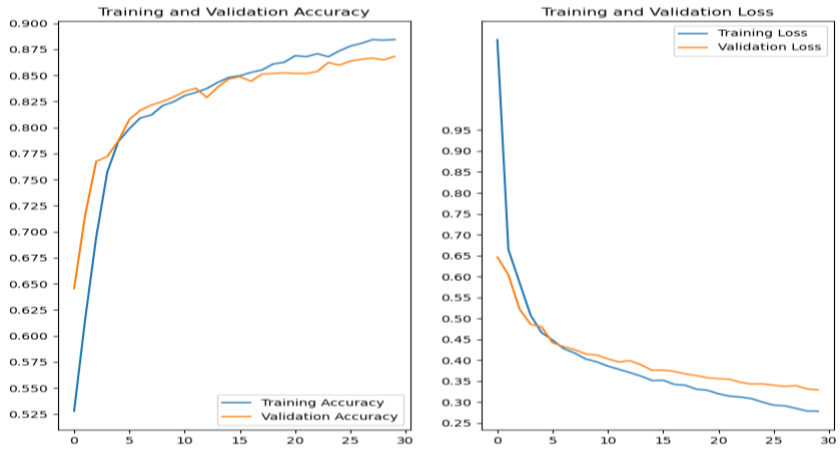


Figure 4.1 Accuracy and loss of ConvNet on training and validation data

each iteration, the time cost is 15s-16s, hence the total training time is approx. 40s on our computer. We trained the ConvNet and the Figure1 shows the performance of the model.

The accuracy converged to 91% on the training dataset and 85% on the test data. We use L1 regularization to avoid overfitting and added learning rate on adam optimization function to help the model converge to higher accuracy and lower loss. We choose L1=0.0001 in the training process. From the plot, it can be seen that the performance of this model is good, it's neither overfitting nor underfitting. Figure 4.2 shows the confusion matrix and Figure 4.3 shows the ROC curve of the ConvNet.

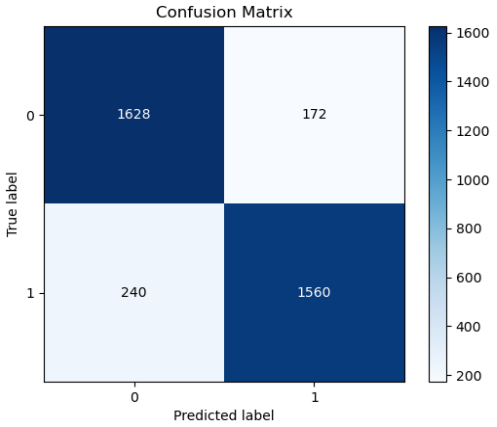


Figure 4.2 Confusion Matrix for prediction using CNN

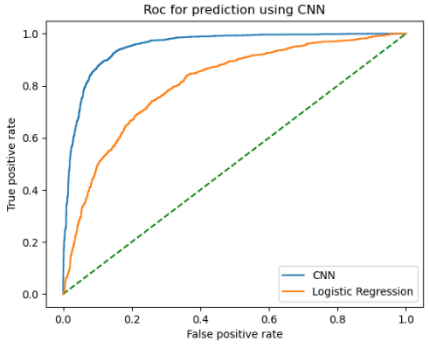


Figure 4.3 ROC curve for prediction using CNN

The confusion matrix paints a clear image of the performance of the ConvNet in each category. The result illustrates our model has a good performance in both two categories. The reason why is that during generating our dataset, we generated the data on each of these categories equally, to avoid the low accuracy caused by the unbalanced data. The result shows the approach works well. This can also be seen from the ROC curve of this model. Table 4.2 gives a clear show of the performance matrices on the training data and test data including precision, recall, and f1-score.

Table 4.2 Performance matrix of training data

Train/ Test	Train				Test			
	precision	**recall**	**f1-score**	**support**	**precision**	**recall**	**f1-score**	**support**
0 (Child)	0.93	0.89	0.91	7200	0.87	0.86	0.87	1800
1 (Adult))	0.89	0.93	0.91	7200	0.86	0.88	0.87	1800
Accuracy	-	-	0.91	14400	-	-	0.87	3600
macro avg	0.91	0.91	0.91	14400	0.87	0.87	0.87	3600
weighted avg	0.91	0.91	0.91	14400	0.87	0.87	0.87	3600

※recall: True Positive Rate

Figure 4.4 shows the performance of the model with the default learning rate for adam optimizer. We found that it is very slow for the model to converge to higher accuracy and lower loss values with default learning rate, and we can see from the plot it is a little overfitting at the end of the training process.

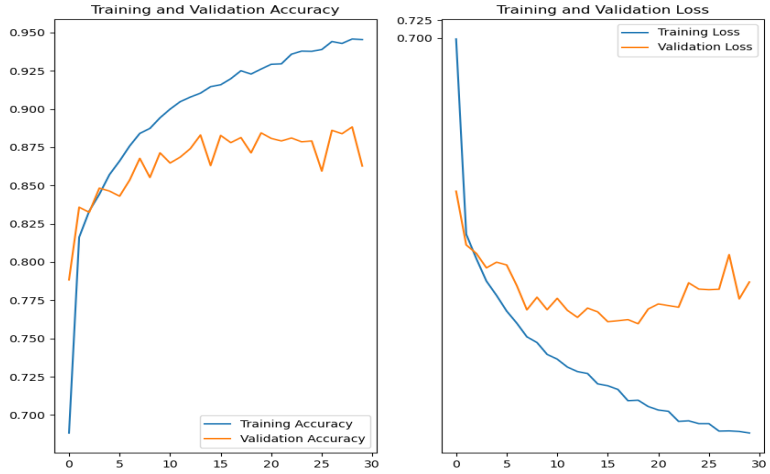


Figure 4.4 Accuracy and loss of ConvNet with default learning rate

II. PCA & Logistic Regression (Baseline)

We choose Logistic Regression with no penalty terms as the baseline model, thus there are no hyperparameters to tune. F_1 score was used as the metrics, whose expression is $F_1 = \frac{2TP}{2TP+FN+FP}$.

Step 1 PCA

Images in the training set were first processed using PCA to reduce dimensionality for faster computation. We set a threshold of 0.99 to find the first k eigenvectors that can cover over 99% of effective features. To do so, a step value of 100 was set to rapidly reach the point near the threshold and then reset the step to 1 to find the exact k that we want. The final ideal k for our training data is 917, whereas the original number of components in each input image is 2304 (48×48). By applying PCA, about 60% of the components were reduced.

Step 2 Model training

In this step, we used the LogisticRegression model provided by sklearn library to train the model. All parameters remained their default values.

Step 3 Evaluation on the testing set

The confusion matrix for the model is shown in figure 4.5 and the F_1 score of the model can then be calculated whose result is 0.73. The accuracy is 0.74. And the ROC curve is plotted in figure 4.6.

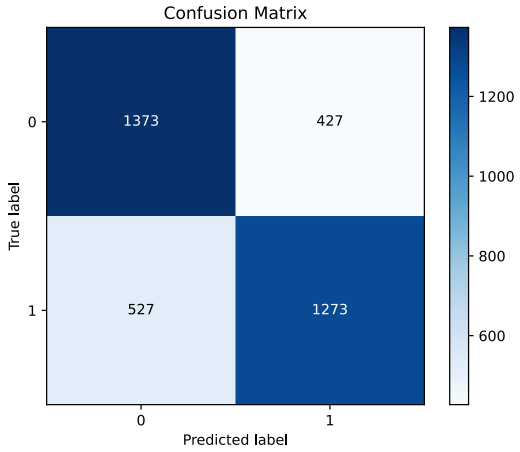
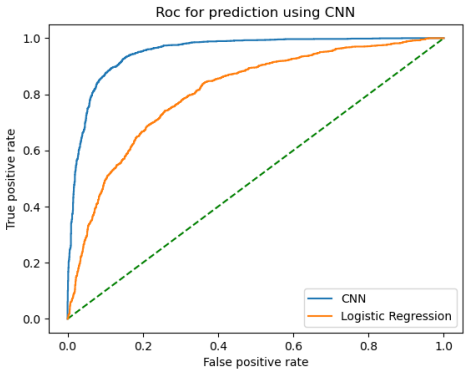


Figure 4.5 Confusion Matrix Figure



4.6 ROC curve

Between the two models we have trained, convolutional neural network achieved more precise prediction results compared with the performance of the baseline model. Figure 4.6 have shown the comparison of ROC between these models clearly. Table 4.3 have shown the comparison of accuracy, loss, f1-score and time cost of each model. We could find the time cost of training CNN models is significantly higher than Logistic regression model since CNNs are more sophisticated in structure. But evidence prove that when it comes to image processing problem, CNN models are more accurate than traditional algorithms.

Table 4.3 Comparison of performances between CNN and Logistic Regression

	Test Accuracy	**Test Loss**	**f1-score**	**Time cost**
CNN	0.93	0.2923	0.91	40s
LR (Baseline)	0.74	1.65	0.73	15s

5 Summary

In this assignment, we have done most steps of machine learning workflow from the data preparation to the training cycle(Choose Feature→Select Model→Train Model→Test Model). During these procedures, we used many classical methods to deal with the raw dataset such as the **Cascade Classifier**, the **PCA** and some other efficient functions developed by ourselves. Also, we trained our CNN model according to the workflow and compare the result to the baseline (Logistic Regression). The result is the same to what we have supposed that the CNN model is superior to the Logistic Regression model. And there are three main reasons for the result. The first one is that CNN can transfer learning through layers, saving inferences, and making new ones on subsequent layers. Second, there is no need for feature extraction before using the algorithm, it is done during training. Last, it recognizes important features.

6 Contributions

Zhipeng Jia

- Code for grabbing and preprocessing the dataset.
- The section I, 2, 5, 6 and 7 of the report.

Zehua Guo

- Code for removing duplicate images in preprocessing.
- the Logistic Regression and the corresponding parts int section 3, 4 of the report.

Yahong Zhu

- Work of manually getting rid of useless images.
- Code for CNN and the corresponding parts in Section 3, 4 of the report.

7 Include a github link

https://github.com/Kirito-JZP/Group_Assignment

Code Appendix

【Preprocessing.py】

```
import cv2 as cv
import os

ORIGINAL_PATH = "C:/Users/Public/Pictures/Original_picture/Children"
DISCERN_UPLOAD_PATH = "C:/Users/Public/Pictures/Preprocess_dataset/children"
CASCADE_CLASSIFIER_PATH = "setting/haarcascade_frontalface_default.xml"
face_detect = cv.CascadeClassifier(CASCADE_CLASSIFIER_PATH)


def face_detect_fun(img, img_path, index):
    # convert the colour RGB image to a grayscale luminance image
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    face = face_detect.detectMultiScale(gray)
    for x, y, w, h in face:
        roi = gray[y:y + h, x:x + w]
        re_roi = cv.resize(roi, (48, 48))
        # rename each picture
        if index < 10:
            str_index = '000' + str(index)
        elif index < 100:
            str_index = '00' + str(index)
        elif index < 1000:
            str_index = '0' + str(index)
        else:
            str_index = str(index)
        save_path = os.path.join(DISCERN_UPLOAD_PATH, "Child_" + str_index + '.jpg')
        cv.imwrite(save_path, re_roi)


def read_img_batch(path, endpoint=None):
    container = []
    index = 0
    for root, dirs, files in os.walk(path):
        for file in files:
            path = os.path.join(root, file)
            input_img = cv.imread(path)
            # convert the input image to grayscale and scale down it to 1/4 of the
original size.
            face_detect_fun(input_img, path, index)
            index += 1
    return container


read_img_batch(ORIGINAL_PATH)
```

【CNN_Classifier.py】

```
from tensorflow import keras
from tensorflow.keras import regularizers
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
import matplotlib.pyplot as plt
import ReadImage as RI
import datetime

ts = datetime.datetime.now()
x_train, y_train, x_test, y_test = RI.make_dataset()
y_train = keras.utils.to_categorical(y_train, 2)
y_test = keras.utils.to_categorical(y_test, 2)
```

```
# Train model
use_saved_model = False
if use_saved_model:
    model = keras.models.load_model("CNN.model")
else:
    model = keras.Sequential()
    model.add(Conv2D(8, (3, 3), padding='same', input_shape=(48, 48, 1),
activation='relu'))
    model.add(Conv2D(8, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

    model.add(Conv2D(16, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(16, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(2,activation='softmax',
kernel_regularizer=regularizers.l1(0.0001)))
    model.compile(loss="categorical_crossentropy", optimizer='adam',
metrics=["accuracy"])
    model.summary()

    batch_size = 64
    epochs = 20
    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_data=(x_test, y_test))
    model.save("CNN.model")
    plt.subplot(121)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.subplot(122)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss'); plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    te = datetime.datetime.now()
    print('time elapsed: {}s'.format((te - ts).total_seconds()))
    plt.show()
```

【Logistic_Regression.py】

```
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, roc_curve, f1_score, log_loss
from scikitplot.metrics import plot_confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
import datetime
import ReadImage as RI


def train_lr(x, y, C=1, penalty='none', solver='lbfgs'):
    lr_model = LogisticRegression(penalty=penalty, solver=solver, max_iter=3000,
C=C).fit(x, y)
```

```
    return lr_model


def cross_validation(model, x, y, cv):
    scores = cross_val_score(model, x, y, cv=cv, scoring='f1')
    model.fit(x, y)
    mean_score = np.array(scores).mean()
    std_dev = np.array(scores).std()
    return (mean_score, std_dev)


def convert_to_vector(imgs):
    ret = []
    for img in imgs:
        ret.append(img.reshape(img.size))
    return np.array(ret)


def get_pca():
    Children_train = "Image/Children_train"
    Adults_train = "Image/Adults_train"
    x_children_train = RI.read_img_batch(Children_train)
    x_adults_train = RI.read_img_batch(Adults_train)
    x_train = np.array(x_children_train + x_adults_train)
    pca = PCA(917)
    pca.fit(convert_to_vector(x_train))
    return pca

def find_dimension(imgs):
    vectorised_x_train = convert_to_vector(imgs)
    n_components = 1
    while True:
        print(n_components)
        pca = PCA(n_components)
        pca.fit_transform(vectorised_x_train)
        flag = False
        if pca.explained_variance_ratio_.sum() > 0.99:
            n_components -= 100
            while True:
                print(n_components)
                pca = PCA(n_components)
                pca.fit_transform(vectorised_x_train)
                if pca.explained_variance_ratio_.sum() > 0.99:
                    flag = True
                    break
                else:
                    n_components += 1
            if flag:
                break
        else:
            n_components += 100
    return n_components


def get_plot_data():
    x_train, y_train, x_test, y_test = RI.make_dataset()

    pca = PCA(917)
    pca.fit(convert_to_vector(x_train))

    reduced_x_train = pca.fit_transform(convert_to_vector(x_train))
    model = train_lr(reduced_x_train, y_train.ravel())
    fpr, tpr, _ = roc_curve(y_test,
model.decision_function(pca.transform(convert_to_vector(x_test))))

    return (fpr, tpr)
```

```
if __name__ == '__main__':
    ts = datetime.datetime.now()
    x_train, y_train, x_test, y_test = RI.make_dataset()

    # find #component
    n_components = find_dimension(x_train)
    pca = PCA(n_components)
    reduced_x_train = pca.fit_transform(convert_to_vector(x_train))
    model = train_lr(reduced_x_train, y_train.ravel())
    score = f1_score(y_test.ravel(),
model.predict(pca.transform(convert_to_vector(x_test))))
    # evaluation on training set
    pred = model.predict(reduced_x_train)
    cm_lr = confusion_matrix(y_train.ravel(),
model.predict(pca.transform(convert_to_vector(x_train))))
    tn, fp, fn, tp = cm_lr.ravel()
    print("(on training set)")
    print("tn: {}, fp: {}, fn: {}, tp: {}".format(tn, fp, fn, tp))
    print("accuracy: {}".format((tn + tp) / (tn + tp + fn + fp)))
    print("f1 score: {}".format(2 * tp / (2 * tp +fn + fp)))
    print("cross-entropy loss: {}".format(log_loss(y_train,
model.predict(pca.transform(convert_to_vector(x_train))))))
    # evaluation on test set
    pred = model.predict(pca.transform(convert_to_vector(x_test)))
    cm_lr = confusion_matrix(y_test.ravel(),
model.predict(pca.transform(convert_to_vector(x_test))))
    tn, fp, fn, tp = cm_lr.ravel()
    print("(on test set)")
    print("tn: {}, fp: {}, fn: {}, tp: {}".format(tn, fp, fn, tp))
    print("accuracy: {}".format((tn + tp) / (tn + tp + fn + fp)))
    print("f1 score: {}".format(2 * tp / (2 * tp +fn + fp)))
    y_cat_0 = y_test
    y_cat_1 = np.abs(y_test - 1)
    y_test_one_hot = np.column_stack((y_cat_0, y_cat_1))
    print("cross-entropy loss: {}".format(log_loss(y_test_one_hot,
model.predict_proba(pca.transform(convert_to_vector(x_test))))))
    plot_confusion_matrix(y_test.ravel(), pred.ravel())
    # ROC curve on test set
    fpr, tpr, _ = roc_curve(y_test,
model.decision_function(pca.transform(convert_to_vector(x_test))))
    plt.figure()
    plt.plot(fpr, tpr)
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.plot([0, 1], [0, 1], color='green', linestyle='--')
    te = datetime.datetime.now()
    print('time elapsed: {}s'.format((te - ts).total_seconds()))

    plt.show()
```

【Draw_Figure.py】

```
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
import numpy as np
import scikitplot

from CNN_Classifier import history,epochs,x_test,x_train,y_test,y_train,model
import logistic_regression as LR

# obtain data from the CNN model
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

# plot accuracy on training set and validation set
plt.figure(figsize=(10, 8))
plt.subplot(1, 2, 1)
plt.yticks(np.arange(0.5, 1.0, step=0.025))
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')


# plot loss of training set and validation set
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

# visualise the confusion matrix for the CNN model
yhat_valid = model.predict_classes(x_test)
scikitplot.metrics.plot_confusion_matrix(np.argmax(y_test, axis=1), yhat_valid)
plt.title('Confusion Matrix for prediction using CNN')
plt.show()

# plot ROC curves for the CNN model and the Logistic Regression model
score = model.predict_proba(x_test)[:,1]
fpr, tpr, threshold = roc_curve(np.argmax(y_test, axis=1), score)
plt.plot(fpr, tpr, label='CNN')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Roc for prediction using CNN')
plt.plot([0, 1], [0, 1], color='green',linestyle='--')

# obtain fpr and tpr of the logistic regression model
fpr, tpr = LR.get_plot_data()
plt.plot(fpr, tpr, label='Logistic Regression')
plt.legend()
plt.show()

# generate report
preds = model.predict(x_train)
y_pred = np.argmax(preds, axis=1)
y_train1 = np.argmax(y_train, axis=1)
print(classification_report(y_train1, y_pred))

preds = model.predict(x_test)
y_pred = np.argmax(preds, axis=1)
y_test1 = np.argmax(y_test, axis=1)
print(classification_report(y_test1, y_pred))
```