Supplementary File 1: CNN binary image classifier .

```
import os
import random from PIL import Image, ImageFilter
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dropout, Conv2D, MaxPooling2D, Flatten, Dense
from keras.optimizers import Adam
from sklearn.metrics import confusion_matrix
import seaborn as sns
import pickle
data_dir1 = "H:\\Qt\\Projects\\SuanZaoRen\\FiguresNew\\Class_0"
data_dir2 = "H:\\Qt\\Projects\\SuanZaoRen\\FiguresNew\\Class_1"
label1 = 0
label2 = 1
def load_data(data_dir, label):
    images = []      labels = []
    for filename in os.listdir(data_dir):
        if filename.endswith(".bmp"):
            image_path = os.path.join(data_dir, filename)
            image = Image.open(image_path).convert('RGB')
            target_size = (299, 299)
            image = image.resize(target_size, Image.LANCZOS)
            images.append(np.array(image))
            labels.append(label)
    return images, labels
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=180,
    width_shift_range=0.5,
    height_shift_range=0.5,
    shear_range=0.7,
    zoom_range=0.7,
    horizontal_flip=True,
    vertical_flip=True,    # Added vertical flip
    fill_mode='nearest',
    channel_shift_range=150.0,    # Added channel shift
    brightness_range=[0.2,1.0]    # Added brightness range
images1, labels1 = load_data(data_dir1, label1)
images2, labels2 = load_data(data_dir2, label2)
combined_images = images1 + images2
combined_labels = labels1 + labels2
combined_data = list(zip(combined_images, combined_labels))
random.shuffle(combined_data)
# Unpack combined data
images, labels = zip(*combined_data)
x_data = np.array(images)
y_data = np.array(labels)
# Define the checkpoint directory to save the models
checkpoint_path =
```

```
"H:\\Qt\\Projects\\SuanZaoRen\\model_checkpoints\\model_epoch_{epoch:02d}.h5"
        # Split into training and validation sets
        x_train, x_val, y_train, y_val = train_test_split(x_data, y_data, test_size=0.1,
random_state=42)
        x_train = x_train / 255.0
        x_val = x_val / 255.0
        image_height, image_width, num_channels = 299, 299, 3
        model = Sequential([
            Conv2D(32, (3, 3), input_shape=(image_height, image_width, num_channels),
activation='relu'),
            Conv2D(32, (3, 3), activation='relu'),
            MaxPooling2D(pool_size=(2, 2)),
            Conv2D(64, (3, 3), activation='relu'),
            Conv2D(64, (3, 3), activation='relu'),
            MaxPooling2D(pool_size=(2, 2)),
            Conv2D(128, (3, 3), activation='relu'),
            Conv2D(128, (3, 3), activation='relu'),
            MaxPooling2D(pool_size=(2, 2)),
            Dropout(0.25),
            Conv2D(256, (3, 3), activation='relu'),
            Conv2D(256, (3, 3), activation='relu'),
            MaxPooling2D(pool_size=(2, 2)),
            Dropout(0.25),
            Conv2D(512, (3, 3), activation='relu'),
            Conv2D(512, (3, 3), activation='relu'),
            MaxPooling2D(pool_size=(2, 2)),
            Dropout(0.25),
            Flatten(),
            Dense(512, activation='relu'),
            Dense(1, activation='sigmoid')
        custom_learning_rate = 0.01
        optimizer = Adam(learning_rate=custom_learning_rate)
        # Compile model
        model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy',
'Precision', 'Recall'])
        # Train model
        batch_size = 16
        epochs = 100
        train_generator = train_datagen.flow(x_train, y_train, batch_size=batch_size)
        history = model.fit(
            #train_generator,
            x_train, y_train,
            steps_per_epoch=len(x_train) // batch_size,
            epochs=epochs,
```

```
            validation_data=(x_val, y_val),
        #test_loss, test_accuracy, test_precision, test_recall = model.evaluate(x_val, y_val,
verbose=2)
        #print(f"Test accuracy: {test_accuracy}")
        model.save('best_model.h5')
        # Save training history to file
        with open('model_history.pkl', 'wb') as file:
            pickle.dump(history.history, file)
        epochs = range(1, len(history.history['accuracy']) + 1)
        # Creating a new figure
        plt.figure(figsize=(12, 6))
        # Plotting accuracy, precision, and recall
        plt.plot(epochs, history.history['accuracy'], 'b-', label='Accuracy')
        plt.plot(epochs, history.history['val_accuracy'], 'b--', label='Validation Accuracy')
        plt.plot(epochs, history.history['precision'], 'g-', label='Precision', alpha=0.7)
        plt.plot(epochs, history.history['val_precision'], 'g--', label='Validation Precision',
alpha=0.7)
        plt.plot(epochs, history.history['recall'], 'y-', label='Recall', alpha=0.7)
        plt.plot(epochs, history.history['val_recall'], 'y--', label='Validation Recall', alpha=0.7)
        # Labels and title
        plt.title('Training and Validation Metrics')
        plt.xlabel('Epochs')
        plt.ylabel('Metrics')
        # Adding legend
        plt.legend(loc='center right')
        plt.show()
        # Confusion Matrix plot
        # Predict the values from the validation dataset
        Y_pred = model.predict(x_val)
        # Convert predictions classes to one hot vectors
        Y_pred_classes = np.argmax(Y_pred, axis=1)
        # Compute the confusion matrix
        confusion_mtx = confusion_matrix(y_val, Y_pred_classes)
        # Plotting
        plt.figure(figsize=(12, 6))
        sns.heatmap(confusion_mtx, annot=True, fmt='d', cmap='Blues')
        plt.ylabel('True Label')
        plt.xlabel('Predicted Label')
        plt.title('Confusion Matrix')
        plt.show()
```