Cars Image recognition

(تقــريــر حول مشروع التعرف على صور السيارات)

يهدف هذا المشروع إلى تصنيف السيارات إلى 3 فئات "sports car", "family car" و "small car" باستخدام شبكة عصبونية باستخدام حزمة TensorFlow. يتم ذلك عن طريق تدريب النموذج باستخدام مجموعة من الصور المصنفة مسبقًا، يتم تقسيمها إلى مجموعة تدريب واختبار. بعد التدريب، يتم استخدام النموذج لتحليل صور جديدة وتصنيفها إلى الفئة الصحيحة.

تم استخدام حزمة TensorFlow لبناء النموذج، حيث تم استخدام طبقات مختلفة مثل التصفية، التجميع، والطبقات الكاملة الاتصال لتدريب النموذج. تم تحديد النموذج باستخدام وظيفة التصنيف متعددة الفئات (softmax)لتحديد الفئة الأكثر احتمالًا للصورة المدخلة.

تم استخدام مكتبة PIL لفتح وتغيير حجم الصور، بينما تم استخدام حزمة NumPy لتحويل الصور إلى مصفوفات وتنسيق البيانات لتتكون من مصفوفات ثنائية الأبعاد. تم استخدام حزمة scikit-learn لتقسيم البيانات إلى مجموعة تدريب واختبار.

تم تحديد الخوارزمية المناسبة لتحسين النموذج باستخدام خوارزميةAdam optimizer ، وتم استخدام دالة الخسارة sparse_categorical_crossentropy لقياس خطأ التصنيف.

تم تدريب النموذج باستخدام 25 دورة تدريب، وتم تقييم أداء النموذج باستخدام دقة التصنيف ودالة الخسارة.

أخيرًا، تم استخدام النموذج المدرب سابقًا لتحليل صور جديدة وتصنيفها إلى الفئة الصحيحة. وتم عرض التسمية النهائية للصورة باستخدام الفهرس الذي تم إنشاؤه للتسميات والفئات.

بشكل عام، يمكن استخدام هذا المشروع لتصنيف السيارات بدقة عالية

الكود التالي بلغة بايثون python :

```
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split
from PIL import Image

# جمع البيانات
```

```
car_photos = ['sport_car.jpg', 'sport_car2.jpg', 'sport_car3.jpg',
'sport_car4.jpg', 'sport_car5.jpg', 'family_car.jpg', 'family_car2.jpg',
'family_car3.jpg', 'family_car4.jpg', 'family_car5.jpg', 'small_car.jpg',
'small_car2.jpg', 'small_car3.jpg', 'small_car4.jpg', 'small_car5.jpg']
labels = [0, 1, 2]
data = []
count = i = int(0)
for photo in car_photos:
    count += 1
    img = Image.open(photo)
    img = img.resize((224, 224))
    img = np.array(img)
    data.append([img, labels[i]])
    if count == 5:
        i += 1
        count = 0

# تقسيم البيانات
train_data, test_data = train_test_split(data, test_size=0.2,
random_state=42)
# x_train تحتوي على صورة التدريب
x_train = np.array([item[0] for item in train_data])
# y_train تحتوي على تصنيف صورة التدريب
y_train = np.array([item[1] for item in train_data])
x_test = np.array([item[0] for item in test_data])
y_test = np.array([item[1] for item in test_data])

# تحديد النموذج
# تسمح بترتيب الطبقات التي تشكل النموذج بترتيب متسلسل
model = tf.keras.models.Sequential([
    # بحجم (3,3)، وتفعيلها باستخدام وظيفة التنشيط (filter) تطبيق تصفية على الصورة، حيث يتم تحديد 32 فلتر
relu.
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                           input_shape=(224, 224, 3)),
    # تقوم بتقليل حجم الصورة عن طريق استخدام تقنية التجميع (pooling)
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    # تقوم بتحويل الصورة من صيغة مصفوفة ثنائية الأبعاد إلى مصفوفة واحدة ذات بُعد واحد.
    tf.keras.layers.Flatten(),
    # هي طبقة كاملة الاتصال (fully connected) حيث يتم تحديد 128 عقدة
    tf.keras.layers.Dense(128, activation='relu'),
    # طبقة كاملة الاتصال تحتوي على 3 عقد بسبب أن هناك 3 فئات لتصنيف الصور
```

```
    tf.keras.layers.Dense(3, activation='softmax')
])

# تدريب النموذج
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=25, validation_data=(x_test, y_test))

# استخدام النموذج
img = Image.open('sport_car_test.jpg')
# img = Image.open('family_car_test.jpg')
# img = Image.open('small_car_test.jpg')
img = img.resize((224, 224))
img = np.array(img)
# تم تطبيع القيم في المصفوفة لتكون قيم بين 0 و 1.
img = img / 255.0
img = img.reshape((1,) + img.shape)
# سابقًا للتنبؤ بفئة الصورة التي تم تحليلها.
pred = model.predict(img)
#  الوظيفة argmax للعثور على الفئة الأكثر احتمالًا للصورة.
class_idx = tf.argmax(pred, axis=1)
class_label = ['sports car', 'family car', 'small car'][class_idx[0]]
print(class_label)
```

output

Epoch 1/25

1/1 [==============================] - 2s 2s/step - loss: 23.0802 - accuracy: 0.3333 - val_loss: 1517.8373 - val_accuracy: 0.3333

Epoch 2/25

1/1 [==============================] - 1s 924ms/step - loss: 1506.1676 - accuracy: 0.4167 - val_loss: 1769.9215 - val_accuracy: 0.3333

Epoch 3/25

1/1 [==============================] - 1s 937ms/step - loss: 1580.5714 - accuracy: 0.3333 - val_loss: 556.4556 - val_accuracy: 0.3333

Epoch 4/25

1/1 [==============================] - 1s 962ms/step - loss: 478.2578 - accuracy: 0.3333 - val_loss: 73.0923 - val_accuracy: 0.3333

Epoch 5/25

1/1 [==============================] - 1s 922ms/step - loss: 39.4209 - accuracy: 0.4167 - val_loss: 126.0174 - val_accuracy: 0.3333

Epoch 6/25

1/1 [==============================] - 1s 966ms/step - loss: 103.1168 - accuracy: 0.3333 - val_loss: 61.2198 - val_accuracy: 0.3333

Epoch 7/25

1/1 [==============================] - 1s 958ms/step - loss: 47.4837 - accuracy: 0.5833 - val_loss: 45.6057 - val_accuracy: 0.3333

Epoch 8/25

1/1 [==============================] - 1s 932ms/step - loss: 14.7713 - accuracy: 0.5833 - val_loss: 36.4977 - val_accuracy: 0.3333

Epoch 9/25

1/1 [==============================] - 1s 961ms/step - loss: 4.7888 - accuracy: 0.6667 - val_loss: 31.8546 - val_accuracy: 0.3333

Epoch 10/25

1/1 [==============================] - 1s 933ms/step - loss: 0.8468 - accuracy: 0.8333 - val_loss: 11.7260 - val_accuracy: 0.0000e+00

Epoch 11/25

1/1 [==============================] - 1s 936ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 12.9064 - val_accuracy: 0.3333

Epoch 12/25

1/1 [==============================] - 1s 931ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 21.2276 - val_accuracy: 0.6667

Epoch 13/25

1/1 [==============================] - 1s 939ms/step - loss: 1.1325e-06 - accuracy: 1.0000 - val_loss: 29.9687 - val_accuracy: 0.6667

Epoch 14/25

1/1 [==============================] - 1s 913ms/step - loss: 0.9357 - accuracy: 0.9167 - val_loss: 26.5076 - val_accuracy: 0.3333

Epoch 15/25

1/1 [==============================] - 1s 955ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 43.2924 - val_accuracy: 0.0000e+00

Epoch 16/25

1/1 [==============================] - 1s 953ms/step - loss: 0.2187 - accuracy: 0.9167 - val_loss: 44.9595 - val_accuracy: 0.3333

Epoch 17/25

1/1 [==============================] - 1s 949ms/step - loss: 6.3578e-07 - accuracy: 1.0000 - val_loss: 51.1318 - val_accuracy: 0.3333

Epoch 18/25

1/1 [==============================] - 1s 946ms/step - loss: 6.8147e-06 - accuracy: 1.0000 - val_loss: 54.0110 - val_accuracy: 0.3333

Epoch 19/25

1/1 [==============================] - 1s 959ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 55.9941 - val_accuracy: 0.3333

Epoch 20/25

1/1 [==============================] - 1s 1s/step - loss: 0.3785 - accuracy: 0.9167 - val_loss: 33.0021 - val_accuracy: 0.3333

Epoch 21/25

1/1 [==============================] - 1s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 26.9908 - val_accuracy: 0.0000e+00

Epoch 22/25

1/1 [==============================] - 1s 999ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 25.7971 - val_accuracy: 0.3333

Epoch 23/25

1/1 [==============================] - 1s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 27.7428 - val_accuracy: 0.3333

Epoch 24/25

1/1 [==============================] - 1s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 29.4038 - val_accuracy: 0.3333

Epoch 25/25

1/1 [==============================] - 1s 952ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 32.9198 - val_accuracy: 0.3333

1/1 [==============================] - 0s 156ms/step

sports car