Car Accident Detection

By: Goulmemei BASSIME Amram

Table of content

Project Overview

This project aims to develop a deep learning model to classify images of cars into two categories: 'accidented' and 'nonaccident'. The model is trained using Convolutional Neural Networks (CNNs) and is capable of distinguishing between images of cars involved in accidents and those that are not.

Dataset Description

The dataset for this project primarilly consisted two folders (images; which was a mixture of accidented images and nonaccidented images in .jpg format, and labels; which was a mixture of those images in .txt format). The dataset was reorganized and now consist two folders :

- accidented: Consist images of cars involved in accidents.
- aonaccidented : Consist images of cars not involved in accidents.

```
accidented_path = 'accidented'
nonaccident_path = 'nonaccident'
```

All images are in ".jpg" format, and are processed to a standard size, normalized before being used for trainning our model.

Model Architecture

The model used in this project is the CNN (Convolutional Neutral Network) with the following architecture :

- Conv2D Layer: 32 filters, kernel size of (3, 3), activation function 'relu'
- MaxPooling2D Layer: pool size of (2, 2)
- Conv2D Layer: 64 filters, kernel size of (3, 3), activation function 'relu'
- MaxPooling2D Layer: pool size of (2, 2)
- Conv2D Layer: 128 filters, kernel size of (3, 3), activation function 'relu'
- MaxPooling2D Layer: pool size of (2, 2)
- Flatten Layer
- Dense Layer: 128 units, activation function 'relu'
- Dropout Layer: dropout rate of 0.5
- Dense Layer: 2 units, activation function 'softmax'

```python
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='softmax')
])

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

# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)

# Train the Model
history = model.fit(datagen.flow(X_train, y_train, batch_size=32), epochs=10, validation_data=(X_test, y_test))
```

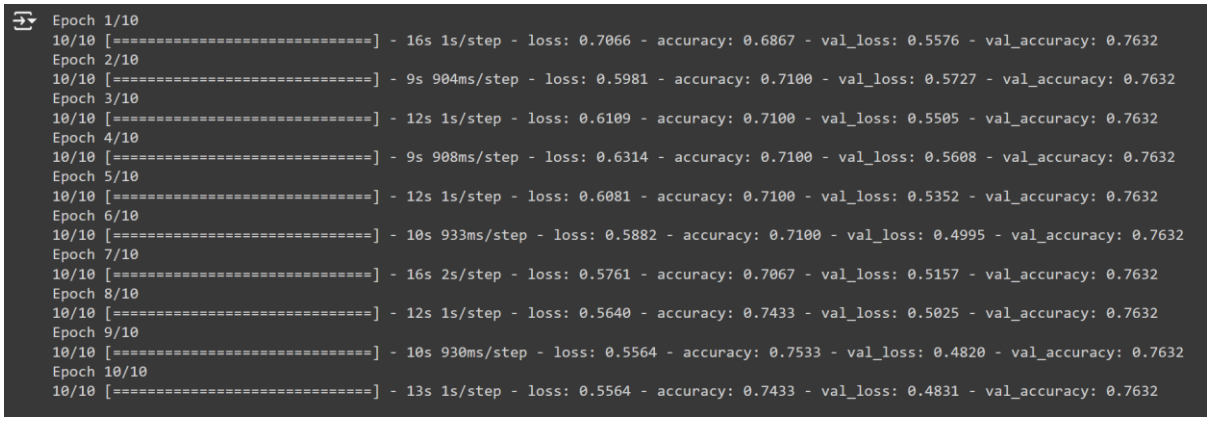
The model is compiled with the Adam optimizer and uses categorical cross-entropy as the loss function. Accuracy is used as the evaluation metric.

Training Process

The model is trained using images from both classes ('accidented' and 'nonaccident'). Data augmentation techniques such as rotation, width shift, height shift, and horizontal flip are used to enhance the training data and improve the model's generalization capability.

The training process involves:

1) Loading and preprocessing the images.
2) Splitting the dataset into training and testing sets.
3) Building the CNN model.
4) Training the model using the training data.
5) Evaluating the model using the testing data.



As seen on this previous image, The epochs was just set at a maximum of 10 (ten). Inorder to train more and more our model, we decided to extend our epoch to 100. This made our value loss (val_loss) to drop down to ***0.2*** at about the ***50th epoch*** and ***accuracy*** and ***val_accuracy*** to increase to about ***0.8.*** Note that in the first epochs, value loss was up to **0.5** which is huge. In the 90th epoch, it became more. The following image illustrate for the 50th epoch :

```
12s 1s/step - loss: 0.3648 - accuracy: 0.8433 - val_loss: 0.2795 - val_accuracy: 0.8553

12s 1s/step - loss: 0.3339 - accuracy: 0.8200 - val_loss: 0.2637 - val_accuracy: 0.8421
```

The following image illustrate for the 90th epoch :

```
11s 1s/step - loss: 0.2922 - accuracy: 0.8600 - val_loss: 0.1816 - val_accuracy: 0.9079
```

The following table shows in more explicit form the improvement of accuracy, loss :

	Loss	Accuracy	Val_loss	Val_accuracy
1st Epoch	0.7066	0.6867	0.5576	0.7632
50th Epoch	0.3648	0.8433	0.2795	0.8553
90th Epoch	0.2922	0.8600	0.1816	0.9079

6) Saving the trained model for future use (in our case, our file was saved under the name "car_accident_detection_model.h5").

Usage

NOTE : The following codes have already being given in a file named "***detect_accident.ipynb***". That file can be directly used to detect accident by just changing the image path in it at every time. These following codes are just to explain how it is processed.

To use the trained model for predicting whether a car image is accidented or nonaccident, open your drive and create a folder called "**dataset**" in it and upload all the files given with this documentation in the folder. Files included in it MUST contain "*test_images*" folder, "*detect_accident.ipynb*" and "*car_accident_detection_model.h5*" follow these steps:

1) Load the Model

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/driv
```

```
[19] %cd /content/drive/MyDrive/dataset
```

```
/content/drive/MyDrive/dataset
```

2) Process the image

```
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
```

```
[21] # Load the trained model
model_path = 'car_accident_detection_model.h5'
model = load_model(model_path)
```

```
[22] # Function to preprocess the image
def preprocess_image(image_path, target_size=(128, 128)):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, target_size)
    img = img / 255.0  # Normalize to [0, 1]
    img = np.expand_dims(img, axis=0)  # Add batch dimension
    return img
```

3) Make a prediction

```python
# Load and preprocess the test image
test_image_path = 'test_images/image6.jpg'  # Replace with your test image path
test_image = preprocess_image(test_image_path)
```

```python
# Make a prediction
prediction = model.predict(test_image)
predicted_class = np.argmax(prediction, axis=1)[0]

# Interpret the prediction
classes = ['nonaccident', 'accidented']
predicted_label = classes[predicted_class]

# Display the result
print(f'The model predicts this image is: {predicted_label}')
```

4) Visualize the result

```python
# Display the test image
plt.imshow(cv2.cvtColor(cv2.imread(test_image_path), cv2.COLOR_BGR2RGB))
plt.title(f'Prediction: {predicted_label}')
plt.axis('off')
plt.show()
```

With this, our images after being tested, accidented images will have the following result:

```
1/1 [==============================] - 0s 45ms/step
The model predicts this image is: accidented
```

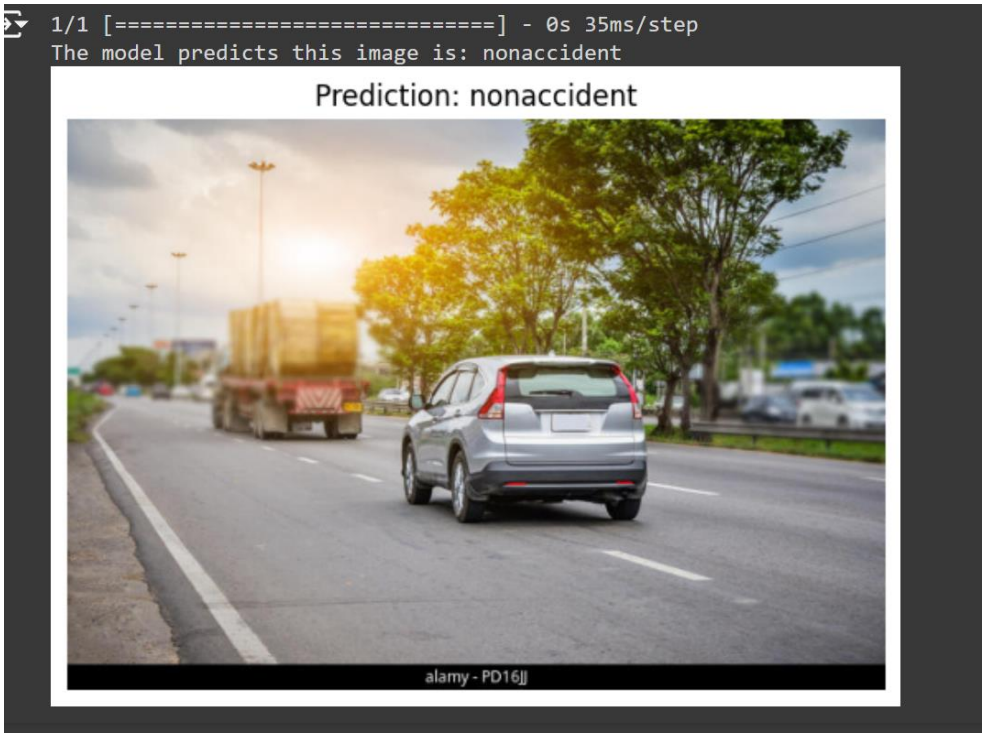
```
1/1 [==============================] - 0s 37ms/step
The model predicts this image is: accidented
```


```
1/1 [==============================] - 0s 33ms/step
The model predicts this image is: accidented
```

Non accidented images will be shown as follows :

```
1/1 [==============================] - 0s 45ms/step
The model predicts this image is: nonaccident
```


```
1/1 [==============================] - 0s 32ms/step
The model predicts this image is: nonaccident
```



Conclusion

This project demonstrates the use of Convolutional Neural Networks for image classification in the context of detecting car accidents. The model can accurately classify images into 'accidented' and 'nonaccident' categories after being trained on a suitable dataset. Further improvements can be made by increasing the dataset size, trying different model architectures, or using transfer learning techniques.

The source code, documentation and presentation and all dataset of this project are available on github and can be accessed via the link :

Primary link - https://github.com/Amrampro/AI-car-accident-detection.git

Second Link - Amrampro/AI-car-accident-detection (github.com)