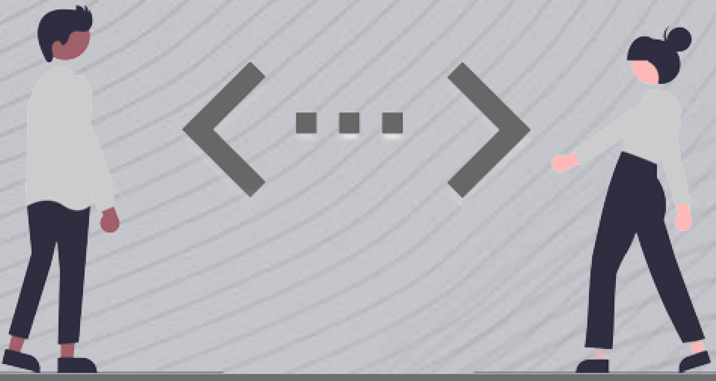
Kingdom of Saudi Arabia
Qassim University
College of Computer
Computer Science

المملكة العربية السعودية
جامعة القصيم
كية الحاسب
قسم علوم الحاسب

Deep Learning Approach to Image Classification of Normal Cars and Military Vehicles

DATA SCIENCE - IT351



Students:
Abdullah Alshami – 412117701
Ragheed Almasry – 412117664
Supervisor:
Dr.Abdurrahman Aloraini

Table of Contents

1. Introduction

Image classification is an important problem in the field of computer vision. With the increasing availability of large amounts of image data, deep learning algorithms have proven to be effective in image classification tasks.

The development of an image that can differentiate between normal cars and military vehicles has numerous potential applications in the field of computer vision.

In this project, we used TensorFlow and Keras to create a convolutional neural network (CNN) capable of accurately classifying military vehicles. The project followed an iterative process of data collection, preprocessing, model training, and enhancement, resulting in a high-performance model capable of accurately classifying military vehicles.

2. Methodology

2.1 Dataset Collection

The initial dataset comprised approximately 8,000 car images, we got from Kaggle. Additionally, a dataset of 3,000 military vehicles images was collected from various sources, including the USA, UK, KSA, Russia, Canada, and Germany. This combined dataset formed the basis of our training data.

2.2 Data Preprocessing

The dataset was preprocessed to remove any unsuitable photos, resulting in a military vehicle dataset of 2,493 images. An imbalance in the dataset was addressed using Python and applying data augmentation techniques, increasing the number of military vehicles samples to 9,972. However, during evaluation, the model faced confusion between military vehicles and old/broken cars. To address this, 1,401 images of old and broken cars were collected, resulting in a revised dataset comprising 9,972 military vehicles and 9,401 normal cars.

2.3 Dataset Splitting

The revised dataset was split into three subsets: training (70%), validation (20%), and testing (10%). The training set was used to train the model, the validation set was used to tune the hyperparameters and prevent overfitting, and the test set was used to evaluate the final performance of the model.

2.4 Model Creation

The model created using Keras and comprised of the following layers:

- Four Conv2D layers (with increasing number of filters: 32, 64, 128, and 256) followed by MaxPooling2D layers.
- BatchNormalization layers were applied after each Conv2D layer. Batch normalization is a technique that normalizes the output of the previous layer, which helps to improve the stability and speed of the training process.
- The features were flattened and passed through two fully connected layers (Dense) with 512 hidden units and a final output unit with a sigmoid activation function. Dense layers are fully connected layers where each neuron is connected to every neuron in the previous layer.
- The model used the Adam optimizer and the BinaryCrossentropy loss function. The Adam optimizer is an adaptive learning rate optimization algorithm that is well-suited for large datasets and high-dimensional parameter spaces. The BinaryCrossentropy loss function is used for binary classification problems, and it calculates the cross-entropy loss between the predicted and actual labels.

2.5 Model Training and Evaluation

The model was trained on the revised dataset for 10 epochs using a batch size of 32.

During training, the model was monitored using TensorBoard, which allowed us to visualize the training process and monitor the accuracy and loss of the model. The validation set was used to tune the hyperparameters and prevent overfitting.

After training, the model was evaluated on the test dataset using Precision, Recall, and BinaryAccuracy metrics. Precision measures the proportion of true positives (predicted military vehicles) among the total number of predicted military vehicles. Recall measures the proportion of true positives among the total number of actual military vehicles.

BinaryAccuracy measures the proportion of correctly classified images out of the total number of images.

2.6 Model Enhancement

The initial version of the model had three Conv2D layers and MaxPooling2D layers, but it was confused between military vehicles and old/broken cars.

To address this, a new layer was added to the model architecture, consisting of a Conv2D layer with 256 filters, a kernel size of (3, 3), ReLU activation, a BatchNormalization layer, and a MaxPooling2D layer.

This enhancement significantly improved the model's performance by enabling it to capture more complex features, increasing the stability and speed of the training process, and reducing the spatial dimensions of the output from the Conv2D layer.

The addition of this layer helped to address the confusion between military vehicles and old/broken cars and led to a substantial improvement in the model's overall performance.

2.7 Model Optimization Summary:

1. Data Collection: Two datasets were collected: 8,000 car images and 3,000 military vehicle images.

- Initial training: Precision = 0.9536709, Recall = 0.9279408, Accuracy = 0.9430044.

2. Data Preprocessing:

- Removed unsuitable photos, resulting in 2,493 military vehicle images.

- Applied data augmentation to address dataset imbalance, increasing military vehicle samples to 9,972.

- Training with augmented dataset: Precision = 0.9921962, Recall = 0.9569892, Accuracy = 0.97552085.

3. Model Training and Evaluation:

- Added 1,401 images of old/broken cars to the dataset (9,972 military vehicles, 9,401 normal cars).

- Improved model performance: Precision = 0.9936709, Recall = 0.9679408, Accuracy = 0.9830044.

4. Model Enhancement:

- Added new layers to the model architecture: Conv2D, ReLU activation, padding='same', BatchNormalization, and MaxPooling2D.

- Enhanced model performance: Precision = 0.99346405, Recall = 0.987013, Accuracy = 0.990625.

3. *Tools and Technologies*

- Python 3: We used Python 3 as the programming language for the project.
- Jupyter Notebook: We used Jupyter Notebook as the development environment for the project.
- Linux OS: We used Linux as the operating system for the project after experiencing issues with Windows.
- Tensorflow: We used the Tensorflow deep learning library to build and train the model.
- OpenCV: We used the OpenCV library for image processing and augmentation.
- Matplotlib: We used the Matplotlib library for visualizing the images and results.
- NumPy: We used the NumPy library for numerical and array operations.

4. Problems we faced

We have faced several problems while working on the project. Here are some of them:

- Time consuming due to weak of GPU and its limitations.
- One of us faced some problems about Jupyter's kernel of Python 3 (ipyKernel).
- During the training process on Windows, the CPU usage was observed to be at 100%, causing performance issues. To address this weakness, we switched to using a Linux system, which resulted in a better experience and slightly faster performance during training. However, we later faced an unknown error that necessitated switching back to Windows.
- Data mixed up and multiple failures while working on the data augmentation.

5. Source Code

Imports

```python
import tensorflow as tf
import os
import cv2
from PIL import Image , ImageEnhance
import numpy as np
from matplotlib import pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
from tensorflow.keras.models import load_model
import random
```

Military Vehicles Augmentations

```python
# Specify the directory containing your images
dir ="C:\\Users\\al7ra\\Downloads\\dataset\\dataset\\data\\RU"
# Create a new directory to store augmented images
output_dir = "RU_augmented"
os.makedirs(output_dir, exist_ok=True)
# List all the image files in the directory
image_files = os.listdir(dir)
# Loop through each image file and apply augmentation
for file in image_files:
    # Load the original image
    original_image = Image.open(os.path.join(dir, file))
    # Perform data augmentation transformations
    augmented_images = []
    # Add original image
    augmented_images.append((original_image, "original"))
    # Rotate image
    rotated_image = original_image.rotate(20)  # Rotation by 20 degrees
    augmented_images.append((rotated_image, "rotation"))
    # Apply random brightness enhancement
    brightness_factor = random.uniform(0.7, 1.3)
    brightness_enhancer = ImageEnhance.Brightness(original_image)
    brightened_image = brightness_enhancer.enhance(brightness_factor)
    augmented_images.append((brightened_image, "brightness"))
    # Apply random contrast enhancement
    contrast_factor = random.uniform(0.8, 1.2)
    contrast_enhancer = ImageEnhance.Contrast(original_image)
    contrasted_image = contrast_enhancer.enhance(contrast_factor)
    augmented_images.append((contrasted_image, "contrast"))
    # Apply random sharpness enhancement
    sharpness_factor = random.uniform(0.5, 1.5)
    sharpness_enhancer = ImageEnhance.Sharpness(original_image)
    sharpened_image = sharpness_enhancer.enhance(sharpness_factor)
    augmented_images.append((sharpened_image, "sharpness"))
    # Save augmented images to the output directory
    for augmented_image, transformation in augmented_images:
        output_filename = f"{transformation}_{file}"
        output_path = os.path.join(output_dir, output_filename)
        augmented_image.save(output_path)
```

Data Clearing

```
data_dir = 'data'
image_exts = ['jpeg','jpg', 'bmp', 'png']
for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir, image_class)):
        image_path = os.path.join(data_dir, image_class, image)
        try:
            # Open the image using Pillow
            img = Image.open(image_path)

            # Get the image type
            image_type = img.format.lower()
            # Check if the image type is in the allowed extensions
            if image_type not in image_exts:
                print(image_type)
                print('Image not in ext list: {}'.format(image_path))
                os.remove(image_path)
        except Exception as e:
            print()
            print(image_type)
            print("not deleted")
            print('Issue with image: {}'.format(image_path))
            print()
            # os.remove(image_path)
```

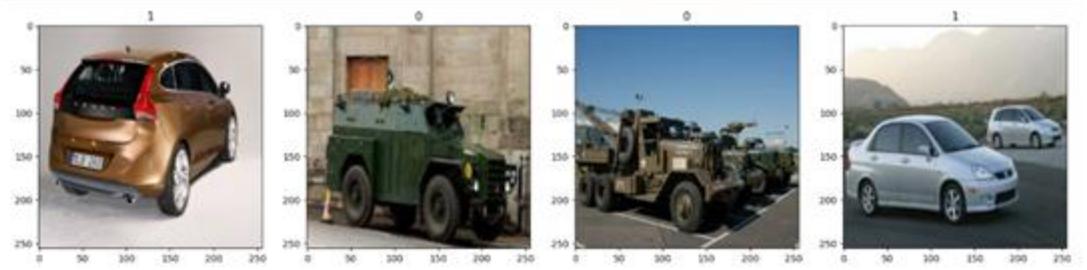
```
data = tf.keras.utils.image_dataset_from_directory('data' , batch_size=32 , image_size=(256,256))
```

Output:

```
Found 19373 files belonging to 2 classes.
```

```
data_iterator = data.as_numpy_iterator()

batch = data_iterator.next()

fig, ax = plt.subplots(ncols=4, figsize=(20,20))

for idx, img in enumerate(batch[0][:4]):

    ax[idx].imshow(img.astype(int))

    ax[idx].title.set_text(batch[1][idx])
```

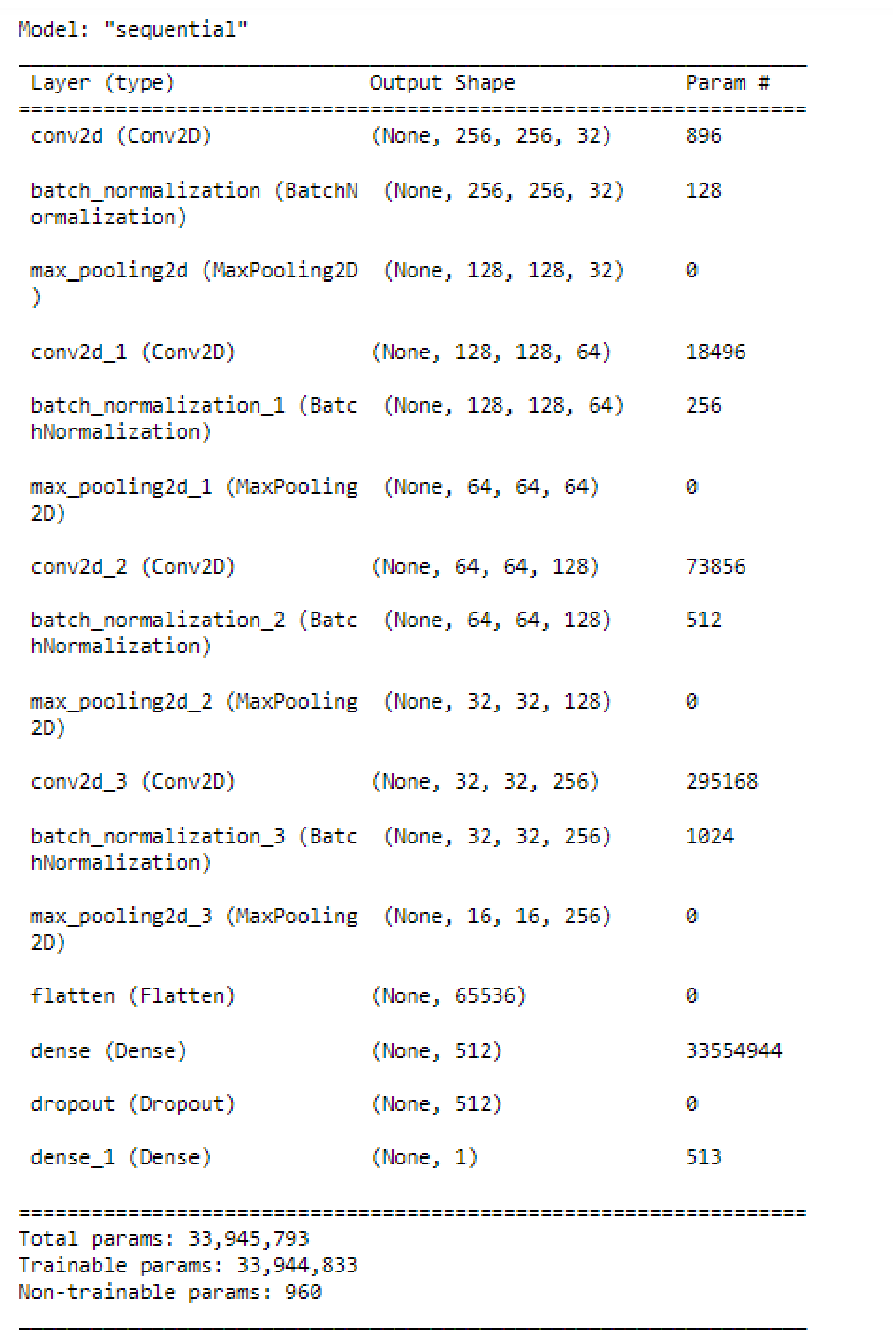
Output:



```
data = data.map(lambda x,y: (x/255, y))
batch = data.as_numpy_iterator().next()
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img)
ax[idx].title.set_text(batch[1][idx])
```

Output :

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(256, 256, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
model.summary()
```



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      896       
                                                                 
 batch_normalization (BatchN  (None, 256, 256, 32)     128       
 ormalization)                                                   
                                                                 
 max_pooling2d (MaxPooling2D  (None, 128, 128, 32)     0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 128, 128, 64)      18496     
                                                                 
 batch_normalization_1 (Batc  (None, 128, 128, 64)     256       
 hNormalization)                                                 
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 64, 64, 64)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 64, 64, 128)       73856     
                                                                 
 batch_normalization_2 (Batc  (None, 64, 64, 128)      512       
 hNormalization)                                                 
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 32, 32, 128)      0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 32, 32, 256)       295168    
                                                                 
 batch_normalization_3 (Batc  (None, 32, 32, 256)      1024      
 hNormalization)                                                 
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 16, 16, 256)      0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 65536)             0         
                                                                 
 dense (Dense)               (None, 512)               33554944  
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 1)                 513       
                                                                 
=================================================================
Total params: 33,945,793
Trainable params: 33,944,833
Non-trainable params: 960
_________________________________________________________________
```

```
logdir='logs'
```

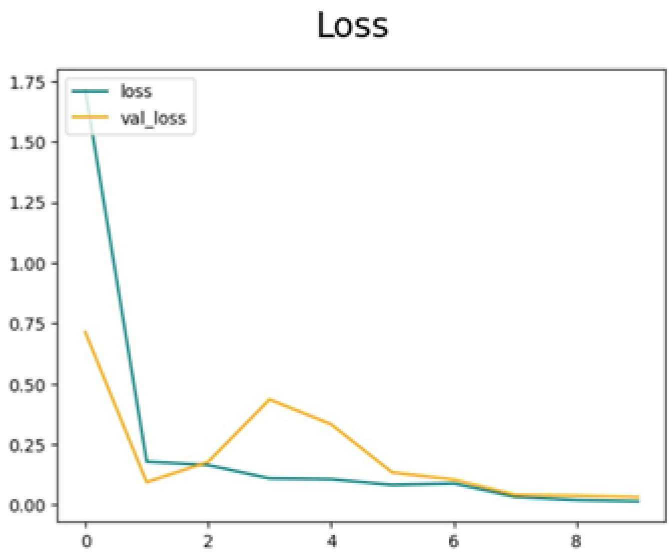
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

```
hist = model.fit(train, epochs=10, validation_data=val, callbacks=[tensorboard_callback])
```

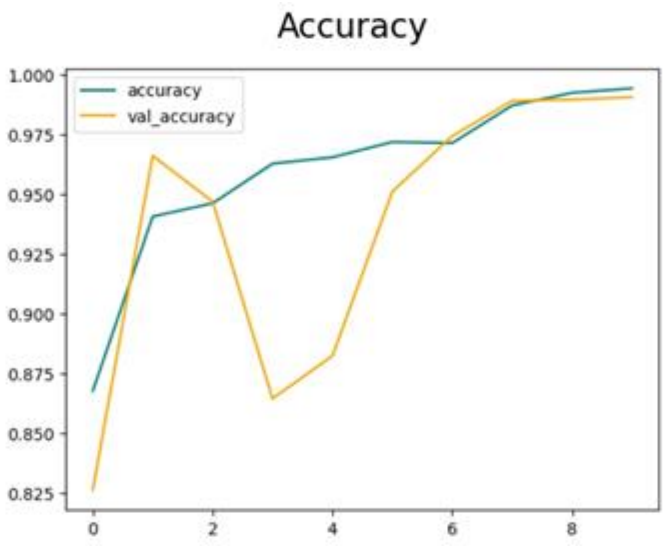
Output:

```
Epoch 1/10
424/424 [==============================] - 1848s 4s/step - loss: 1.7167 - accuracy: 0.8676 - val_loss:
0.7146 - val_accuracy: 0.8262 - lr: 0.0010
Epoch 2/10
424/424 [==============================] - 1915s 5s/step - loss: 0.1791 - accuracy: 0.9405 - val_loss:
0.0942 - val_accuracy: 0.9659 - lr: 0.0010
Epoch 3/10
424/424 [==============================] - 1755s 4s/step - loss: 0.1650 - accuracy: 0.9460 - val_loss:
0.1778 - val_accuracy: 0.9468 - lr: 0.0010
Epoch 4/10
424/424 [==============================] - 1755s 4s/step - loss: 0.1099 - accuracy: 0.9626 - val_loss:
0.4360 - val_accuracy: 0.8644 - lr: 0.0010
Epoch 5/10
424/424 [==============================] - 1758s 4s/step - loss: 0.1067 - accuracy: 0.9653 - val_loss:
0.3339 - val_accuracy: 0.8822 - lr: 0.0010
Epoch 6/10
424/424 [==============================] - 1749s 4s/step - loss: 0.0827 - accuracy: 0.9717 - val_loss:
0.1343 - val_accuracy: 0.9509 - lr: 0.0010
Epoch 7/10
424/424 [==============================] - 1746s 4s/step - loss: 0.0893 - accuracy: 0.9713 - val_loss:
0.1049 - val_accuracy: 0.9742 - lr: 0.0010
Epoch 8/10
424/424 [==============================] - 1751s 4s/step - loss: 0.0334 - accuracy: 0.9870 - val_loss:
0.0418 - val_accuracy: 0.9889 - lr: 2.0000e-04
Epoch 9/10
424/424 [==============================] - 1746s 4s/step - loss: 0.0202 - accuracy: 0.9923 - val_loss:
0.0386 - val_accuracy: 0.9894 - lr: 2.0000e-04
Epoch 10/10
424/424 [==============================] - 1752s 4s/step - loss: 0.0153 - accuracy: 0.9941 - val_loss:
0.0323 - val_accuracy: 0.9904 - lr: 2.0000e-04
```

```
fig = plt.figure()
plt.plot(hist.history['loss'], color='teal', label='loss')
plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



```
fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



```
pre = Precision()
re = Recall()
acc = BinaryAccuracy()
for batch in test.as_numpy_iterator():
    X, y = batch
    yhat = model.predict(X)
    pre.update_state(y, yhat)
    re.update_state(y, yhat)
acc.update_state(y, yhat)
print("Precision:" ,pre.result().numpy(),", Recall :", re.result().numpy(),", Accuracy :",
acc.result().numpy())
```

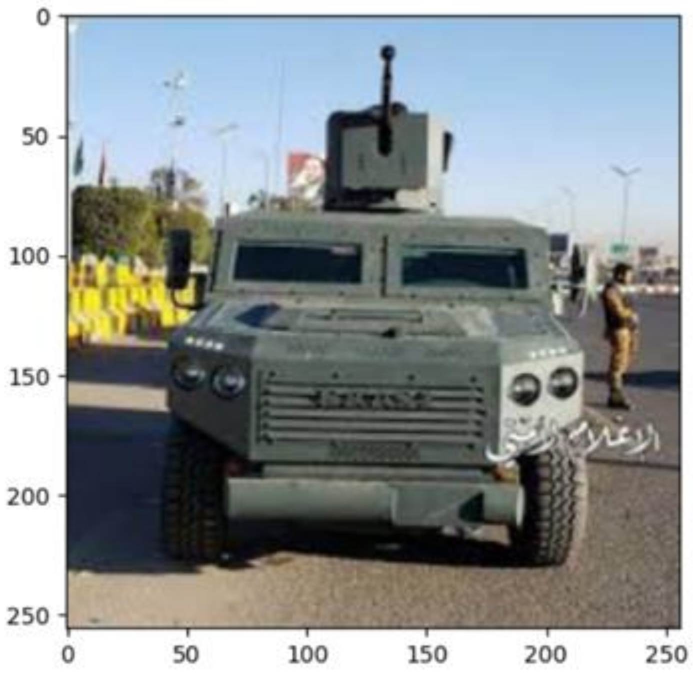
Output :

Precision: 0.99346405 , Recall : 0.987013 , Accuracy : 0.990625

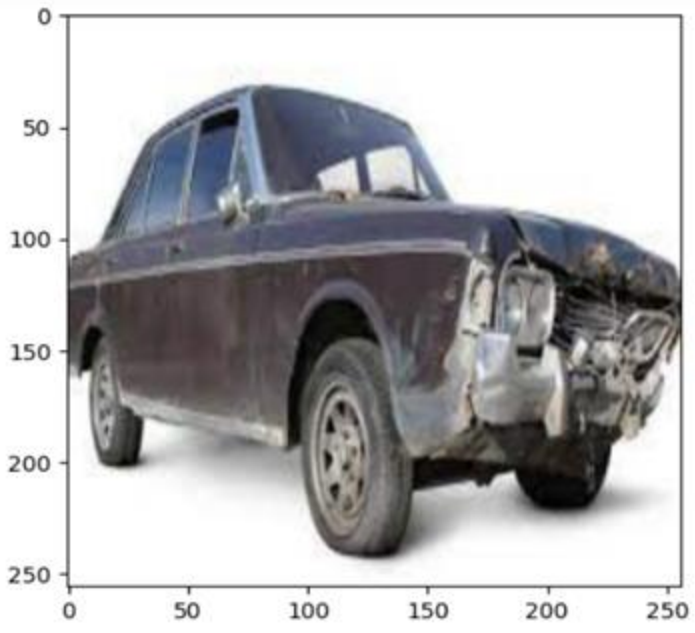
```
directory_path = "C:\\Users\\al7ra\\Downloads\\dataset\\dataset\\test"
img = cv2.imread(directory_path+'\\5.jpeg')
resized_img = cv2.resize(img, (256, 256))
plt.imshow(cv2.cvtColor(resized_img, cv2.COLOR_BGR2RGB))
plt.show()
yhat = model.predict(np.expand_dims(resized_img/255, 0))
if yhat > 0.5:
    print(int(yhat[0][0]*100), "% chance that it is a normal car")
else:
    print(100-int(yhat[0][0]*100), "% chance that it is a military vehicle")
```

```
1/1 [==============================] - 0s 172ms/step
100 % chance that it is a normal car
```

6. Examples

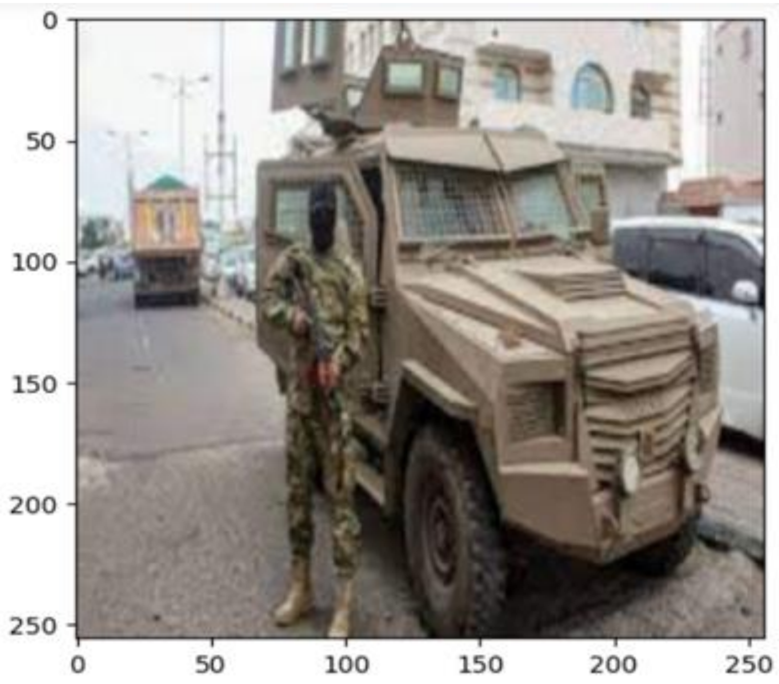


```
1/1 [==============================] - 0s 178ms/step
100 % chance that it is a military car
```

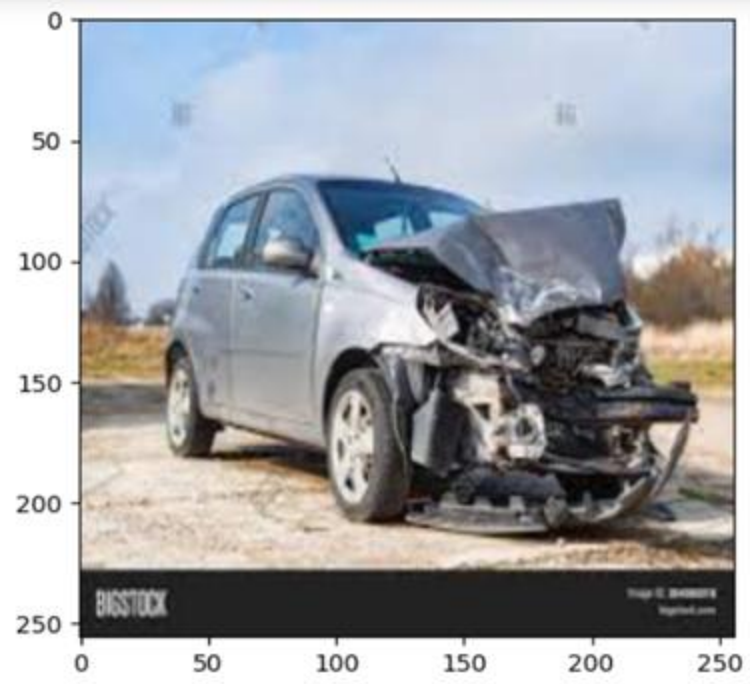


```
1/1 [==============================] - 0s 180ms/step
81 % chance that it is a normal car
```

Examples



```
1/1 [==============================] - 0s 169ms/step
99 % chance that it is a military car
```



```
1/1 [==============================] - 0s 166ms/step
94 % chance that it is a normal car
```

7. Results

The final model achieved high precision, recall, and accuracy, indicating its effectiveness in accurately classifying military vehicles while distinguishing them from normal and old/broken cars. The enhanced model achieved the following evaluation metrics:

- Precision: 0.99346405
- Recall: 0.987013
- Accuracy: 0.990625

These evaluation metrics indicate that the model has a high precision, which means that it is highly accurate in identifying military vehicles. The recall value indicates that the model has a high sensitivity in identifying military vehicles, and the accuracy value shows that the model is highly accurate in classifying both military and normal cars.

8. Conclusion

In conclusion, this project demonstrated the effectiveness of TensorFlow and Keras in developing an image classifier capable of distinguishing between normal cars and military vehicles. Through the iterative process of data collection, preprocessing, model training, and enhancement, we achieved substantial improvements in the model's performance. The final model exhibited high precision, recall, and accuracy, indicating its effectiveness in accurately classifying military vehicles while distinguishing them from normal and old/broken cars.

This project can be further expanded by exploring different architectures, optimizing hyperparameters, and using different datasets.

9. References

- analyticsvidhya.com
- medium.com
- datacamp.com
- cnvrg.io
- tensorflow.org
- keras.io
- en.wikipedia.org
- en.wikipedia.org