Car classification

Deep learning 1000 ML

Agenda

- Introduction
- Read Data
- Data Augmentation
- Resize all images
- Load images
- Splitting data
- Standardization
- Label encoding
- Creating pretrained model
- Testing the model

Introduction

The problem is to identify the type of car through the camera, and this project can be used in many things, such as identifying the type of car in entering and exiting private places and other projects

The solution We used an DL model to identify the type of car

Reading Data

library

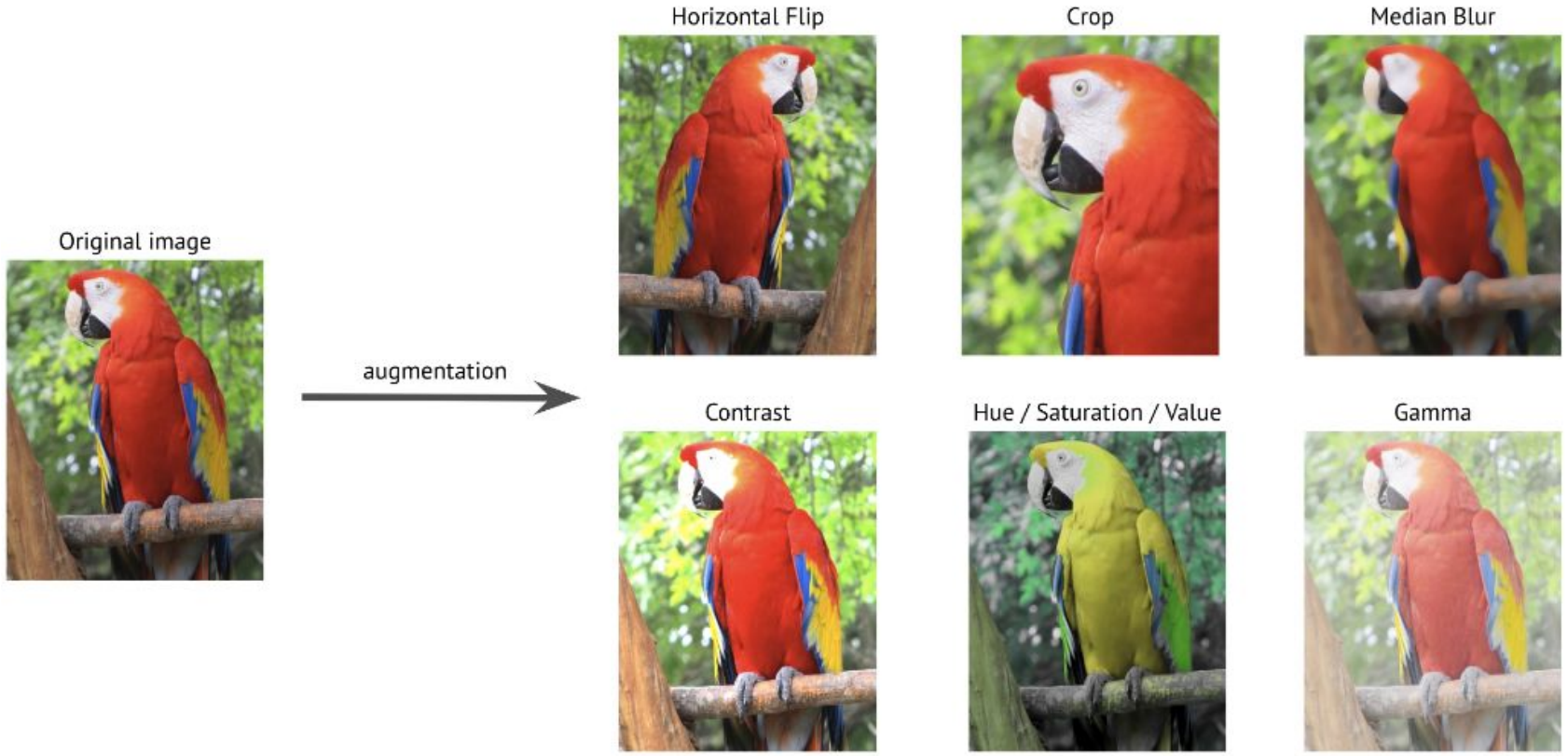
```python
# import necessary libraries
import os
import glob
import cv2
import random
import albumentations as A
import numpy as np
from sklearn.preprocessing import LabelEncoder,StandardScaler
import joblib
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.resnet50 import ResNet50
from keras.applications.mobilenet import MobileNet
from keras.layers import GlobalAveragePooling2D, Dropout, Dense
from tensorflow.keras.layers import Dense,Flatten
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, CSVLogger
import time

```

Read data

```python
for subdir in os.listdir(root_dir):
    subdir = os.path.join(root_dir, subdir)
    if os.path.isdir(subdir):
        images = glob.glob(os.path.join(subdir, '*.jpg'))
        for image in images:
            # Load the image
            img = cv2.imread(image)
            # Resize the image
            img_resized = cv2.resize(img, (width, height))
            # Save the resized image
            cv2.imwrite(image, img_resized)
duration = time.time() - start_time
```

Data Augmentation



Data Augmentation and resizing all images

Data Augmentation Resize all images

```python
def augment_images(root_dir, width, height):
    """
    Augment the images in the given directory and write the augmented images to the same directory.
    """
    print(f"\nAugmenting images started..")
    start_time = time.time()
    max_images = 0
    for subdir in os.listdir(root_dir):
        subdir = os.path.join(root_dir, subdir)
        if os.path.isdir(subdir):

            images = glob.glob(os.path.join(subdir, '*.jpg'))
            max_images = max(max_images, len(images))


    for subdir in os.listdir(root_dir):
        subdir = os.path.join(root_dir, subdir)
        if os.path.isdir(subdir):
            images = glob.glob(os.path.join(subdir, '*.jpg'))
            while len(images) < max_images:
                label = os.path.basename(subdir)
                img = cv2.imread(random.sample(images,1)[0])
                if img is None:
                    continue
                else:

                    img_resized = cv2.resize(img, (width, height))
                    img_aug = apply_augmentations()(image=img_resized)['image']
                    cv2.imwrite(f'{subdir}/augmented_{len(images)}.jpg',img_aug)
                    images.append(f'{subdir}/augmented_{len(images)}.jpg')
    for subdir in os.listdir(root_dir):
        subdir = os.path.join(root_dir, subdir)
        if os.path.isdir(subdir):
            images = glob.glob(os.path.join(subdir, '*.jpg'))
            for image in images:
                # Load the image
                img = cv2.imread(image)
                # Resize the image
                img_resized = cv2.resize(img, (width, height))
                # Save the resized image
                cv2.imwrite(image, img_resized)
    duration = time.time() - start_time
    print(f"apply_augmentations() took {round(duration, 2)} seconds")
```

Augmentation function

```python
def apply_augmentations():
    """
    Define a set of image augmentations and return an instance of the augmentation pipeline.
    """
    aug = A.Compose([
        A.HorizontalFlip(),
        A.Rotate(limit=15, border_mode=cv2.BORDER_CONSTANT),
        A.GaussNoise(var_limit=(0, 30)),
        A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2),
        A.RandomGamma(gamma_limit=(80, 120)),
        A.RandomScale(scale_limit=(0.7, 1.3)),
        A.RandomCrop(height=192, width=192, p=0.25)
    ])

    return aug
```

Loading and splitting data

Loading data

```
def load_data(root_dir):
    """
    Load the images from the given directory and return them as numpy arrays of images and labels.
    """
    print(f"\nLoading data started..")
    start_time = time.time()
    X = []
    y = []
    for subdir in os.listdir(root_dir):
        subdir = os.path.join(root_dir, subdir)
        if os.path.isdir(subdir):
            images = glob.glob(os.path.join(subdir, '*.jpg'))
            for image in images:
                img = cv2.imread(image)
                X.append(img)
                label = os.path.basename(subdir)
                y.append(label)
    X = np.array(X)
    y = np.array(y)
    duration = time.time() - start_time
    print(f"load_data() took {round(duration, 2)} seconds")
    return X, y
```

Splitting data

```
def split_data(X, y):
    # Split the data into training and testing sets
    print(f"\nSplitting data started..")
    start_time = time.time()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
    duration = time.time() - start_time
    print(f"split_data() took {round(duration, 2)} seconds")
    return X_train, X_test, y_train, y_test
```

Standardization

Standard scaler

```
def scale_data(X_train, X_test):
    # Reshape the data to 2D for scaling
    print(f"\nScaling data started..")
    start_time = time.time()
    X_train = np.reshape(X_train, (X_train.shape[0], -1))
    X_test = np.reshape(X_test, (X_test.shape[0], -1))

    # Scale the data using StandardScaler
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)

    # Save the scaler to file
    scaler_filename = "scaler.save"
    joblib.dump(scaler, scaler_filename)

    # Scale the test data using the fitted scaler
    X_test = scaler.transform(X_test)

    # Reshape the data back to 3D
    X_train = np.reshape(X_train, (X_train.shape[0], 224, 224, 3))
    X_test = np.reshape(X_test, (X_test.shape[0], 224, 224, 3))
    duration = time.time() - start_time
    print(f"scale_data() took {round(duration, 2)} seconds")
    return X_train, X_test
```

Label encoding (using one hot encoding)

```
def encode_labels(y):
    """
    Encode the given labels and return the encoded labels.
    """
    print(f"\nEncoding labels started..")
    start_time = time.time()
    le = LabelEncoder()
    y = le.fit_transform(y)
    y = to_categorical(y, num_classes=20)
    labelEncoder_filename = "labelEncoder.save"
    joblib.dump(le, labelEncoder_filename)
    duration = time.time() - start_time
    print(f"encode_labels() took {round(duration, 2)} seconds")
    return y


def load_label_encode():
    # Load label encoder from file
    le = joblib.load("labelEncoder.save")
    return le
```

Model creation

Creating pretrained model

```
def create_model():
    """
    Load pre-trained MobileNet model and freeze its layers.
    Add additional layers to the model.
    Compile and return the model.
    """
    print(f"\nCreating model started..")
    start_time = time.time()
    base_model = MobileNet(weights='imagenet', include_top=False, input_shape=(224,224,3))
    base_model.trainable = False

#base_model = ResNet50(include_top=False, input_shape=(224, 224, 3), weights = 'imagenet')
#for layer in base_model.layers:
 #   layer.trainable = False
    #model = ResNet50(include_top=False, input_shape=(224, 224, 3), weights = 'imagenet')
# mark loaded layers as not trainable
    #for layer in model.layers[:50]:
        #layer.trainable = False

    #for layer in model.layers[50:]:
        #layer.trainable = True

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dropout(0.5),
        Dense(1024, activation='relu'),
        Dropout(0.5),
        Dense(1024, activation='relu'),
        Dropout(0.5),
        Dense(20, activation='softmax')
    ])
```

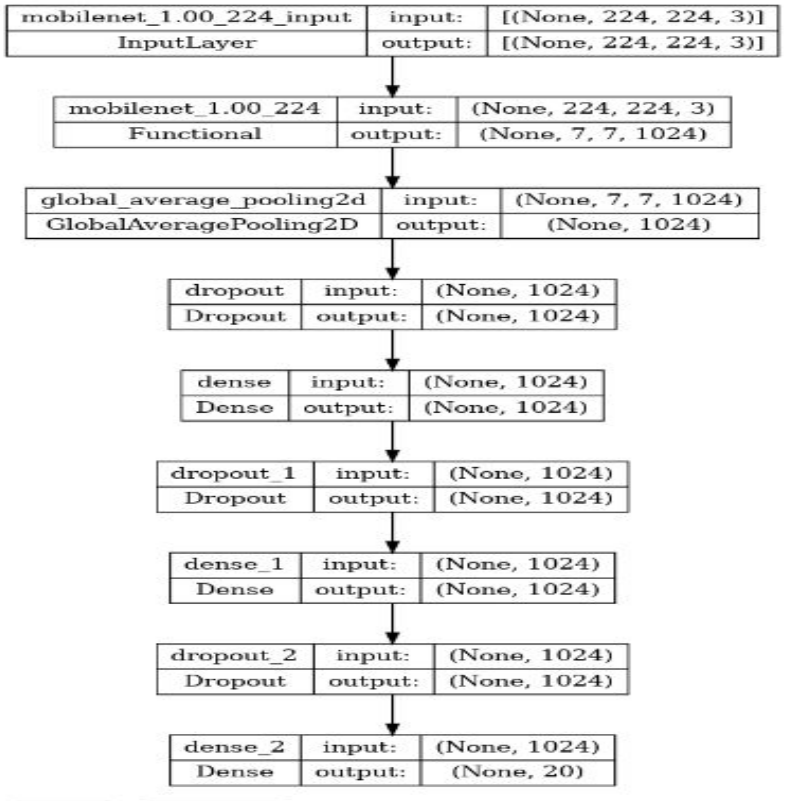
Fitting to our data

```
def train_model(model, X_train, y_train, X_test, y_test):
    """
    Train the model on the given data.
    """
    earlystop=EarlyStopping(patience=25)
    filepath = "model_class.h5"
    checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,save_best_only=True, mode='max')
    log_fname = 'model_log.csv'
    csv_logger = CSVLogger(filename=log_fname,separator=',',append=False)
    callbacks_list = [checkpoint, csv_logger,earlystop]
    print(f"\nTraining model started..")
    start_time = time.time()
    model.fit(X_train, y_train, epochs=220, batch_size=100, validation_data=(X_test, y_test),callbacks=[callbacks_list])
    duration = time.time() - start_time

    print(f"train_model() took {round(duration, 2)} seconds")
```

Model architecture

```
[6]:
1 import tensorflow as tf
2 tf.keras.utils.plot_model(model,show_shapes=True)
```



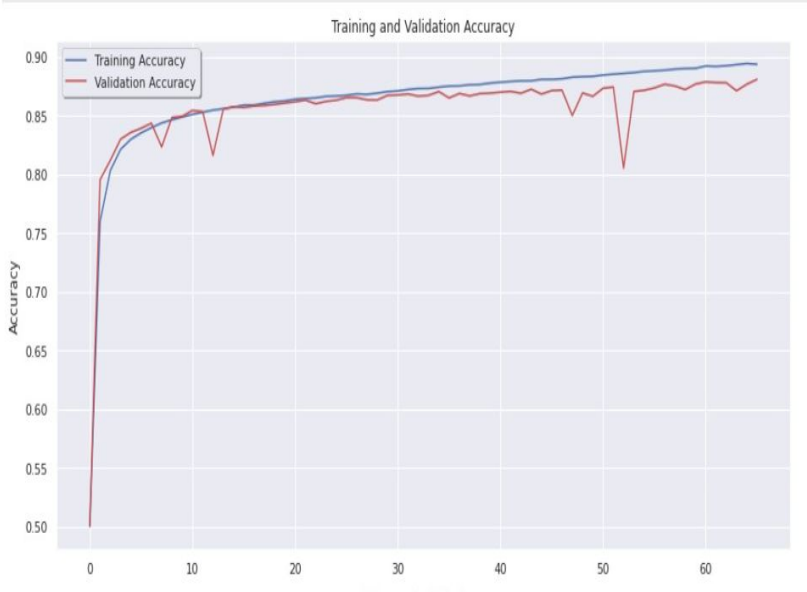
Model Training

After 200 epochs, we reached validated accuracy of 92%

```
loss: 0.1752 - accuracy: 0.9440 - val_loss: 0.3435 - val_accuracy: 0.9222

loss: 0.1980 - accuracy: 0.9371 - val_loss: 0.3460 - val_accuracy: 0.9208
```

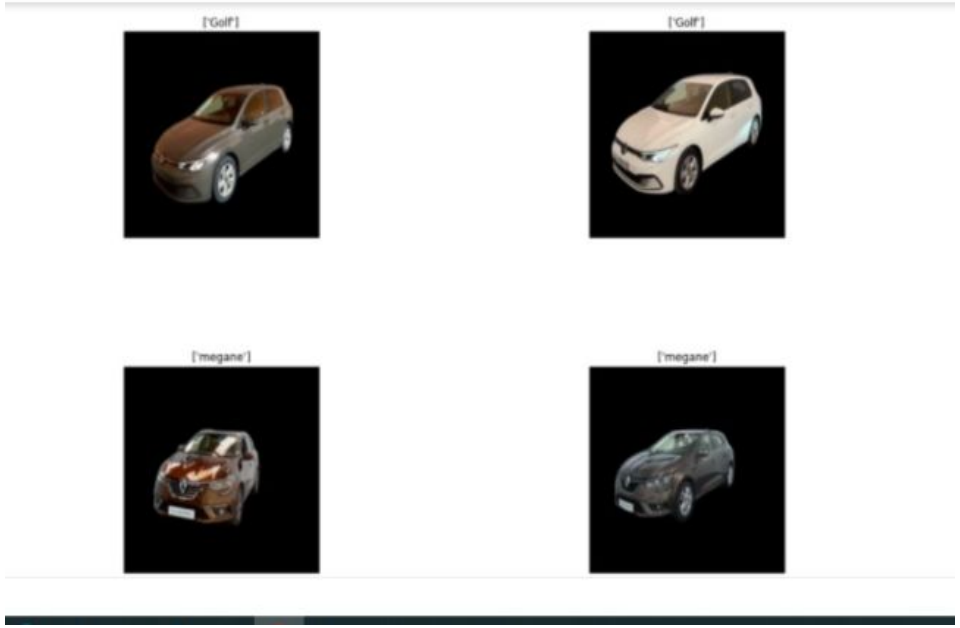
Model evaluation: train/val accuracy plot



Testing the model

Prediction using web app

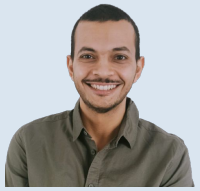
Prediction



Summary

Summary: our project is a multiple classification project to classify the type of the car , we used the transfer learning model (mobile net) and edit the classifier layers we get accuracy 92.5% , also we make a web bag to be as a user interface to take the car image from the user and classify it.

Meet our team



Mohammad Saleh

Mostafa Hussein fathi

Laila Ahmed Khalifa

Mohammad Motasim

Mahmoud Ali

Thank you

Dr/ Hisham Assem