In [1]:

```
import tensorflow as tf
from keras.applications.vgg16 import VGG16
from keras.applications.resnet import ResNet50
import os
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import pandas as pd
import matplotlib.pyplot as plt
```

```
WARNING:tensorflow:From C:\Users\Pikkis\AppData\Local\Programs\Python\Python310\lib\site
-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is d
eprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

## Here prepare the folder if does not exist

In [2]:

```
# you need the current working directory NB: works both windows and linux
current_working_directory = os.getcwd()
current_working_directory = os.path.dirname(current_working_directory)

if not os.path.exists(f"{current_working_directory}/Datasets"):
    os.makedirs(f"{current_working_directory}/Datasets")

print(f"[DATASET] PUT THE DATASET here: {current_working_directory}/Datasets")
```

```
[DATASET] PUT THE DATASET here: C:\Users\Pikkis/Datasets
```

In [3]:

```
# get the directory where I want to download the dataset
path_of_dataset = os.path.join(*['..', current_working_directory, 'Datasets', 'Most_Stol
print(f"[DIR] The directory of the current dataset is {path_of_dataset}")
```

```
[DIR] The directory of the current dataset is C:\Users\Pikkis\Datasets\Most_Stolen_Cars
```

## Data prep

In [4]:

```
# here let s do some functions that we can re-use also for other assignment
def load_the_data_and_the_labels(data_set_path: str, target_size: tuple or None = None):
    """
    This function help you to load the data dynamically
    :param data_set_path: (str) put the path created in the previous cell (is the datase
    :param target_size: (tuple) the desired size of the images
    :return:
        - array of images
        - array with labels
        - list of labels name (this is used for better visualization)
    """
    try:
        dataset, labels, name_of_the_labels = list(), list(), list()
        # let s loop here and we try to discover how many class we have
        for class_number, class_name in enumerate(os.listdir(data_set_path)):
            full_path_the_data = os.path.join(*[data_set_path, class_name])
            print(f"[WALK] I am walking into {full_path_the_data}")

            # add the list to nam _list
            name_of_the_labels.append(class_name)

            for single_image in os.listdir(f"{full_path_the_data}"):
```

```
                full_path_to_image = os.path.join(*[full_path_the_data, single_image])

                # add the class number
                labels.append(class_number)

                if target_size is None:
                    # let s load the image
                    image = tf.keras.utils.load_img(full_path_to_image)
                else:
                    image = tf.keras.utils.load_img(full_path_to_image, target_size=targ

                # transform PIL object in image
                image = tf.keras.utils.img_to_array(image)

                # add the image to the ds list
                dataset.append(image)

        return np.array(dataset, dtype='uint8'), np.array(labels, dtype='int'), name_of_
    except Exception as ex:
        print(f"[EXCEPTION] load the data and the labels throws exceptions {ex}")
```

Load the data

```
a. Target size: (112, 112, 3)
b. if for some reason your pc crash saying Out of Memory reduce half the
target size
```

In [5]:

```
# here
data = load_the_data_and_the_labels(path_of_dataset, (112,112,3))
```

```
[WALK] I am walking into C:\Users\Pikkis\Datasets\Most_Stolen_Cars\chevrolet_impala_2008
[WALK] I am walking into C:\Users\Pikkis\Datasets\Most_Stolen_Cars\chevrolet_silverado_2
004
[WALK] I am walking into C:\Users\Pikkis\Datasets\Most_Stolen_Cars\dodge_ram_2001
[WALK] I am walking into C:\Users\Pikkis\Datasets\Most_Stolen_Cars\ford_f150_2006
[WALK] I am walking into C:\Users\Pikkis\Datasets\Most_Stolen_Cars\gmc_sierra_2012
[WALK] I am walking into C:\Users\Pikkis\Datasets\Most_Stolen_Cars\honda_accord_1997
[WALK] I am walking into C:\Users\Pikkis\Datasets\Most_Stolen_Cars\honda_civic_1998
[WALK] I am walking into C:\Users\Pikkis\Datasets\Most_Stolen_Cars\nissan_altima_2014
[WALK] I am walking into C:\Users\Pikkis\Datasets\Most_Stolen_Cars\toyota_camry_2014
[WALK] I am walking into C:\Users\Pikkis\Datasets\Most_Stolen_Cars\toyota_corolla_2013
```

## normalize the data here

In [6]:

```
# do it here
mean = np.mean(data[0])
std = np.std(data[0])
n_data = [(d-mean)/std for d in data[0]]

data_arr = np.asarray(n_data)
```

## Convert the data to one hot encoding (use the sklearn function)

In [7]:

```
# here we have to one hot encode the labes
def make_the_one_hot_encoding(labels_to_transform):
    try:
        enc = OneHotEncoder(handle_unknown='ignore')
        # this is a trick to figure the array as 2d array instead of list
```

```
        temp = np.reshape(labels_to_transform, (-1, 1))
        labels_to_transform = enc.fit_transform(temp).toarray()
        print(f'[ONE HOT ENCODING] Labels are one-hot-encoded: {(labels_to_transform.sum
        return labels_to_transform
    except Exception as ex:
        print(f"[EXCEPTION] Make the one hot encoding throws exception {ex}")
```

In [8]:
```
# do it here
labels = make_the_one_hot_encoding(data[1])
```

```
[ONE HOT ENCODING] Labels are one-hot-encoded: True
```

# split the data in train set and test set

```
    a. use 0.3 as split factor
```

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(data_arr, labels, test_size=0.3, ran
```

## Create a CNN with the following characteristics

```
        a. Input layer
        b. As base model use VGG16:
            i. Weights: imagenet
            ii. Include_top: False
            iii. Input_shape the target shape described in point 1.
        c. Add a flatten layer
        d. Add a Dense layer with 512 units and a dropout layer with 0.2
    unit.
        e. Add a Dense layer with 256 units and a dropout layer with 0.2
    unit.
        f. Add the final classifier with the correct number of units and the
    suitable activation.
```

alt text

In [10]:
```
# do it here

# Initializing the model
model = tf.keras.Sequential()

# Creating the VGG16 model, it contains an input layer
vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=(112,112,3))

# Adding the base VGG16
model.add(vgg16_model)

# Adding a flatten layer
model.add(tf.keras.layers.Flatten())

# Adding a dense layer with 512 units
model.add(tf.keras.layers.Dense(512))

# Adding a dropout layer
model.add(tf.keras.layers.Dropout(0.2))

ense layer with 512 units
```

```
model.add(tf.keras.layers.Dense(256))

# Adding a dropout layer
model.add(tf.keras.layers.Dropout(0.2))

# Adding a final classifier with our classes (10)
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# Let's visualize our model
model.summary()
```

```
WARNING:tensorflow:From C:\Users\Pikkis\AppData\Local\Programs\Python\Python310\lib\site
-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please
use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\Pikkis\AppData\Local\Programs\Python\Python310\lib\site
-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is depr
ecated. Please use tf.nn.max_pool2d instead.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 3, 3, 512)         14714688

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 512)               2359808

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 10)                2570

=================================================================
Total params: 17208394 (65.64 MB)
Trainable params: 17208394 (65.64 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

## Set the layer block5_conv2, block5_conv3, block5_pool trainable

> Important: you can make a function when you create a CNN within the option of make layers trainable or not is up to you!

In [11]:

```
#do it here
# Setting only the three required layers to trainable
for layer in vgg16_model.layers:
    layer.trainable = False
    if layer.name in ["block5_conv2", "block5_conv3", "block5_pool"]:
        layer.trainable = True
```

## Train the model

a. set the batch size 32 (if your PC go Out of memory lower this number half)

b. set epochs to 15

In [12]:
```
# do it here

# We first need to compile the model. Adam is the best general optimizer and since we ha
# I'll use the categorical crossentropy loss function.
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3), loss=tf.keras.loss

model.fit(X_train, y_train, epochs=15, batch_size=32)
```

Loading [MathJax]/extensions/Safe.js

```
Epoch 1/15
WARNING:tensorflow:From C:\Users\Pikkis\AppData\Local\Programs\Python\Python310\lib\site
-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is depre
cated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Pikkis\AppData\Local\Programs\Python\Python310\lib\site
-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outsid
e_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions i
nstead.

130/130 [==============================] - 56s 420ms/step - loss: 2.3025 - accuracy: 0.1
407
Epoch 2/15
130/130 [==============================] - 53s 410ms/step - loss: 2.2323 - accuracy: 0.1
426
Epoch 3/15
130/130 [==============================] - 52s 399ms/step - loss: 2.2321 - accuracy: 0.1
501
Epoch 4/15
130/130 [==============================] - 52s 398ms/step - loss: 2.2331 - accuracy: 0.1
387
Epoch 5/15
130/130 [==============================] - 52s 399ms/step - loss: 2.2333 - accuracy: 0.1
453
Epoch 6/15
130/130 [==============================] - 52s 397ms/step - loss: 2.2340 - accuracy: 0.1
409
Epoch 7/15
130/130 [==============================] - 51s 396ms/step - loss: 2.2337 - accuracy: 0.1
433
Epoch 8/15
130/130 [==============================] - 52s 397ms/step - loss: 2.2348 - accuracy: 0.1
327
Epoch 9/15
130/130 [==============================] - 52s 396ms/step - loss: 2.2330 - accuracy: 0.1
445
Epoch 10/15
130/130 [==============================] - 52s 397ms/step - loss: 2.2329 - accuracy: 0.1
436
Epoch 11/15
130/130 [==============================] - 52s 398ms/step - loss: 2.2328 - accuracy: 0.1
421
Epoch 12/15
130/130 [==============================] - 52s 397ms/step - loss: 2.2319 - accuracy: 0.1
489
Epoch 13/15
130/130 [==============================] - 52s 403ms/step - loss: 2.2318 - accuracy: 0.1
448
Epoch 14/15
130/130 [==============================] - 57s 438ms/step - loss: 2.2321 - accuracy: 0.1
407
Epoch 15/15
130/130 [==============================] - 57s 437ms/step - loss: 2.2329 - accuracy: 0.1
462
```

Out[12]:
```
<keras.src.callbacks.History at 0x1d59ce4bd00>
```

evaluate the model and record the accuracy score.

In [13]:
```
# do it here
```

```
scores = model.evaluate(X_test, y_test)
```

```
56/56 [==============================] - 19s 338ms/step - loss: 2.2179 - accuracy: 0.153
9
```

## Load again the CNN and set all the base model layers to not trainable.

In [14]:
```
# here

# Initializing the model
model2 = tf.keras.Sequential()

# Creating the VGG16 model, it contains an input layer
vgg16_model2 = VGG16(weights='imagenet', include_top=False, input_shape=(112,112,3))

# Setting all base model layers to non-trainable
for layer in vgg16_model2.layers:
    layer.trainable = False

# Adding the base VGG16
model2.add(vgg16_model2)

# Adding a flatten layer
model2.add(tf.keras.layers.Flatten())

# Adding a dense layer with 512 units
model2.add(tf.keras.layers.Dense(512))

# Adding a dropout layer
model2.add(tf.keras.layers.Dropout(0.2))

# Adding a dense layer with 512 units
model2.add(tf.keras.layers.Dense(256))

# Adding a dropout layer
model2.add(tf.keras.layers.Dropout(0.2))

# Adding a final classifier with our classes (10)
model2.add(tf.keras.layers.Dense(10, activation='softmax'))

# Let's visualize our model
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 vgg16 (Functional)          (None, 3, 3, 512)         14714688  
                                                                 
 flatten_1 (Flatten)         (None, 4608)              0         
                                                                 
 dense_3 (Dense)             (None, 512)               2359808   
                                                                 
 dropout_2 (Dropout)         (None, 512)               0         
                                                                 
 dense_4 (Dense)             (None, 256)               131328    
                                                                 
 dropout_3 (Dropout)         (None, 256)               0         
                                                                 
 dense_5 (Dense)             (None, 10)                2570      
                                                                 
=================================================================
Total params: 17208394 (65.64 MB)
Trainable params: 2493706 (9.51 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

## Repeat the train and evaluation steps

In [15]:

```
# here

model2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3), loss=tf.keras.los

model2.fit(X_train, y_train, epochs=15, batch_size=32)
```

Loading [MathJax]/extensions/Safe.js

```
Epoch 1/15
130/130 [==============================] - 49s 368ms/step - loss: 2.5300 - accuracy: 0.4
182
Epoch 2/15
130/130 [==============================] - 48s 368ms/step - loss: 1.2558 - accuracy: 0.6
205
Epoch 3/15
130/130 [==============================] - 47s 362ms/step - loss: 0.8876 - accuracy: 0.7
237
Epoch 4/15
130/130 [==============================] - 47s 365ms/step - loss: 0.6837 - accuracy: 0.7
892
Epoch 5/15
130/130 [==============================] - 48s 371ms/step - loss: 0.5701 - accuracy: 0.8
339
Epoch 6/15
130/130 [==============================] - 48s 366ms/step - loss: 0.5567 - accuracy: 0.8
359
Epoch 7/15
130/130 [==============================] - 47s 364ms/step - loss: 0.4278 - accuracy: 0.8
760
Epoch 8/15
130/130 [==============================] - 47s 359ms/step - loss: 0.4036 - accuracy: 0.8
891
Epoch 9/15
130/130 [==============================] - 47s 360ms/step - loss: 0.3568 - accuracy: 0.9
002
Epoch 10/15
130/130 [==============================] - 48s 372ms/step - loss: 0.2593 - accuracy: 0.9
285
Epoch 11/15
130/130 [==============================] - 48s 368ms/step - loss: 0.1394 - accuracy: 0.9
550
Epoch 12/15
130/130 [==============================] - 49s 376ms/step - loss: 0.1649 - accuracy: 0.9
517
Epoch 13/15
130/130 [==============================] - 48s 373ms/step - loss: 0.1902 - accuracy: 0.9
434
Epoch 14/15
130/130 [==============================] - 47s 359ms/step - loss: 0.2785 - accuracy: 0.9
243
Epoch 15/15
130/130 [==============================] - 48s 368ms/step - loss: 0.4380 - accuracy: 0.9
002
```

Out[15]:
```
<keras.src.callbacks.History at 0x1d5a6441540>
```

In [16]:
```
# Evaluating our model
scores = model2.evaluate(X_test, y_test)
```

```
56/56 [==============================] - 20s 349ms/step - loss: 4.2766 - accuracy: 0.574
4
```

## What happen? Why?

We got a much more accurate result. This is due to VGG 16 being pre-trained. When we trained it ourselves a bit, it then performed worse as opposed to the more intricate training it had been through performed by its creators.

Make and visualize some predictions.

In [17]:

```python
# here

# I'm reusing the visualization function from last time
def show_some_prediction(number_of_subplot, test_set, predictions, name_of_the_labels):
    for i in range(number_of_subplot):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(test_set[i].astype('uint8'))
        plt.title(f'{name_of_the_labels[predictions[i]]}')
        plt.axis("off")
    plt.subplots_adjust(left=0.1,
                        bottom=0.1,
                        right=0.9,
                        top=0.9,
                        wspace=0.6,
                        hspace=0.6)
    plt.show()
```

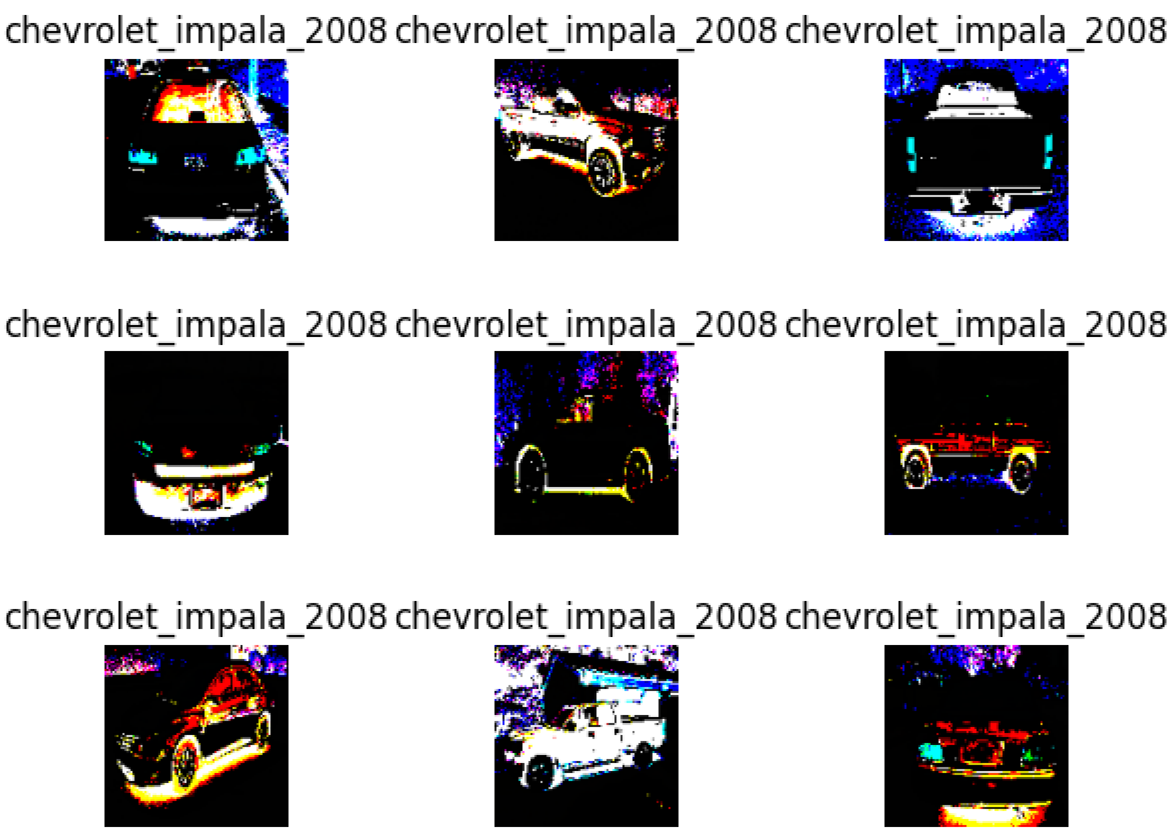
In [18]:

```python
# Predictions for the first one

predictions = model.predict(X_test).astype(int).tolist()
```

```
56/56 [==============================] - 20s 349ms/step
```

In [19]:

```python
real_pred = []

for xs in predictions:
    for x in xs:
        real_pred.append(x)
show_some_prediction(9, X_test, real_pred, data[2])
```



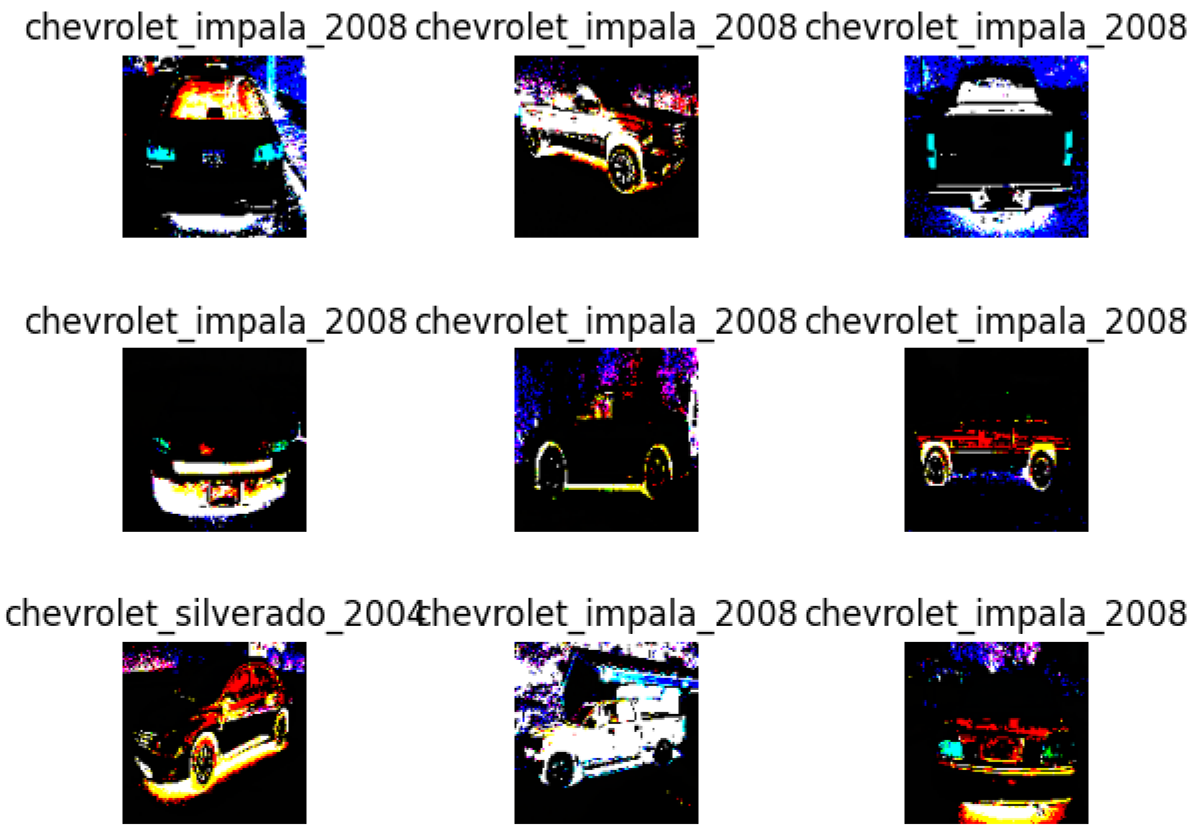

In [20]:

```python
# Predictions for the second one
```

```
predictions2 = model2.predict(X_test).astype(int).tolist()
```

```
56/56 [==============================] - 20s 353ms/step
```

In [21]:
```
real_pred = []

for xs in predictions2:
    for x in xs:
        real_pred.append(x)
show_some_prediction(9, X_test, real_pred, data[2])
```




As we can see, the lower left picture got a different prediction with the original untrained model, possibly the correct one due to increased accuracy.

In [ ]: