Final Project: Vehicle Classification

CS156: Machine Learning for Science and Profit
Minerva Schools at KGI

Albion Krasniqi
December 2020

Table of content:

Problem Description

According to the Used Car Market Report, in 2019, around 103 million used cars were sold, and more than 70% of those sales used online resources through different phases of purchasing. (Grand View Research, 2020). The role of online sites in auto marketplaces increases; thus, they must provide a better experience for the consumers. One of the most common challenges that online sites face today is fraud posts (Yerra, 2019). Usually, such posts contain inconsistencies, e.g., the images they upload do not correspond to the car information.

Using advanced machine learning techniques appropriate for image recognition, we can help solve the challenge mentioned above. In this project, we will build a vehicle recognition predictive model, and its goal will be to classify a car's make and model based on an input image.

Dataset and pre-processing

To build such a model, we will use The Stanford Cars Dataset, an extensive collection of vehicle images. It consists of 16,185 total images labeled with 196 classes based on the car's Make/Model/Year (e.g., Mercedes C class 2009). The data is split into 8,144 training images and 8,041 testing images (roughly 50-50% train-test split), so each class of car has around 40 images in the training set and the same amount in the testing set as well.

Given that humans took images in our dataset, there is no specific angle or order on how pictures were taken. There is high variability in images in the dataset, and working with such a dataset can be tricky. To avoid such problems, we performed data augmentation, which is techniques that increase the diversity of the dataset by applying random (and realistic) transformations (Tensorflow, 2020). Also, for all images we selected the standart size of 224x224x3.

Figure 1. Performing data augmentation on images

Solution Specifications

To build the model, we decided to use Convolutional Neural Networks since they are known to be one of the most effective types of neural networks for image classification (Sorokina, 2017). Next section, we will build different models to classify car dataset and then evaluate their performance.

Costumized CNN

First, I tried to customize a convolutional neural network (model) from scratch; I build a model with 3 Convolutional layers and two dense layers.

Transfer Learning

Transfer learning is a process where you take an existing trained model. In this project, we used two different models MobileNet V2 and EfficientNet B1.

MobileNet V2

MobileNet-v2 is a convolutional neural network that contains 53 layers. This network has been trained on more than a million images from the ImageNet database, and it can classify objects into more than 1000 categories. This CNN has been optimized to perform well on mobile devices (Sandler et al, n.d.).

EfficientNet B1

EfficientNet is among the most efficient models; this network has also been pre-trained on the ImageNet database. In comparison with other CNNs, EfficientNet uses compound scaling, which uniformly scales all dimensions of depth/width/resolution while maintaining the balance (Tan Ming, 2019). That reduces the dimensions of the classification problems significantly.

For all of the models above, we set the last layer to have 196 neurons, which corresponds to the number of car classes. As a loss function, we chose categorical_crossentropy since we are trying to classify categorical data. We added a small learning rate since we wanted to explore more and not get stuck in a suboptimal solution.

Testing and Analysis

After training the models, we had to test them on unseen data (testing data) and see how well they were performing to predict car classes. Based on their performance, it turned out that our customized CNN yielded the poorest results despite having a vast number of parameters; its accuracy was around 2% in both training and testing set. The MobileNet V2 performed better; in the testing set, it achieved an accuracy above 40%. Lastly, as expected, the EfficientNet B1 was the best performing model; it reached a validation accuracy of 71% (see graph below). It is also essential to note that EfficientNet B1, because of compound scaling, took much less time during the training process than other ones.

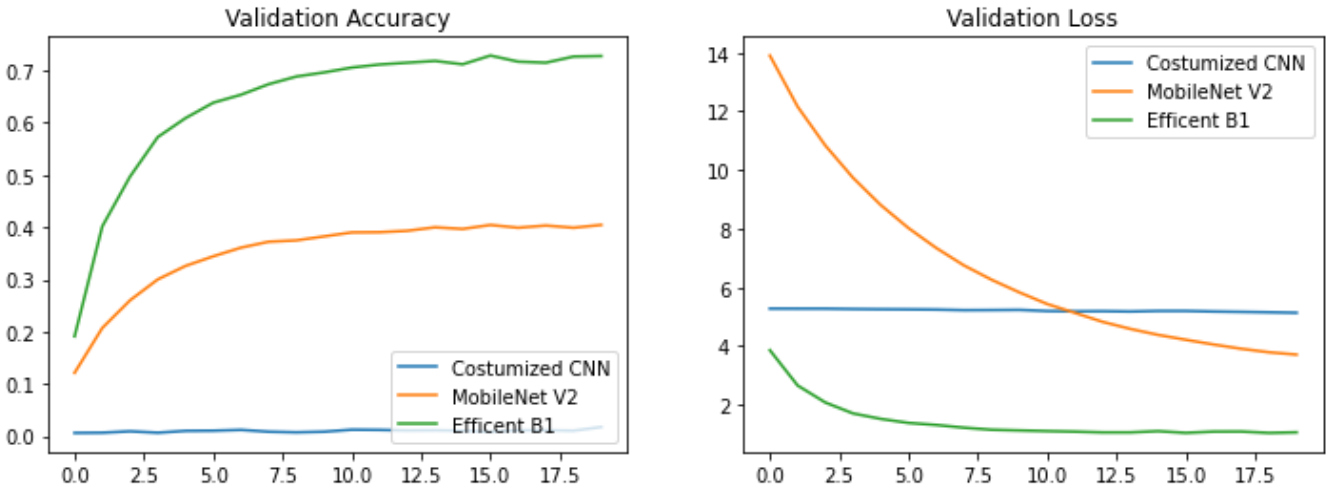


Figure 2. The validation accuracy and loss of models

In addition to testing accuracy, we calculated the precision, recall, and F1-score for each model (see table 1). However, any other metric leads to similar conclusions. Thus, we can conclude that the EfficientNet B1 model surpassed others in terms of both performance and cost.

Table 1. Summary of model performance

Model	Accuracy	Precison	Recall	F-1 score
Costumized CNN	0.018	0.02	0.018	0.019
MobileNet V1	0.40	0.40	0.41	0.40
EfficentNet B1	0.71	0.72	0.71	0.71

Lastly, we used the EfficientNet B1 model to make some predictions for some random images chosen from the testing set (see figure 3).

Figure 3. Making prediction with EfficientNet B1 model

As you can see from figure 3, the EfficientNet B1 model performs relatively well (it classifies most of the cars correctly). While in cases of misclassification, usually cars are very similar to each other. As you can see in figure 3, the bottom-left vehicle was predicted to be a Toyota but a different class from the original one.

References

Grand View Research. (2020). *Used Car Market Size, Share & Trends Analysis Report By Vehicle Type (Hybrid, Conventional, Electric), By Vendor Type, By Fuel Type, By Size, By Region, By Sales Channel, And Segment Forecasts, 2020 - 2027*. From Grand View Research: https://www.grandviewresearch.com/industry-analysis/used-car-market

Kaggle. (n.d.). *Stanford Car Dataset by classes folder*. From https://www.kaggle.com/jutrera/stanford-car-dataset-by-classes-folder

Sandler et al. (n.d.). *MobileNetV2*. From paperswithcode: https://paperswithcode.com/method/mobilenetv2

Sorokina, K. (2017). *Image Classification with Convolutional Neural Networks*. From Medium: https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8

Tan Ming, Q. L. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. From https://arxiv.org/pdf/1905.11946.pdf

Tensorflow. (2020). *Data augmentation*. From Tensorflow: https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8

Yerra, B. (2019). *Classifying Car Images Using Features Extracted from Pretrained Neural Networks*. From Bhanu Yerra's Blog: https://mlbhanuyerra.github.io/2019-10-31-ClassifyingCarImagesByVehileType/

Appendix

Link to .ipynb version of notebook or see the code below:

https://gist.github.com/AlbionKransiqi/7215e4fe966f751c8d9212bc880abaaf

cs156-final-project-vehicle-classification

December 18, 2020

1 Stanford Car Dataset: Vehicle Classification

Importing necessary libraries

[ ]:
```
import numpy as np
import pandas as pd
from PIL import Image
from glob import glob
import matplotlib.pyplot as plt

## MobileNetV2
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense,␣
↪BatchNormalization, Lambda
from tensorflow.keras.models import Model
from tensorflow.keras import regularizers
from tensorflow.keras.layers import Dense, Flatten, Input, Conv1D, Conv2D,␣
↪MaxPooling2D
from tensorflow.keras import layers

## Efficent_B1
import efficientnet.keras as efn
from keras import Sequential
from keras.applications import ResNet50
```

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

```
Mounted at /content/drive
```

2 Data Pre-processing

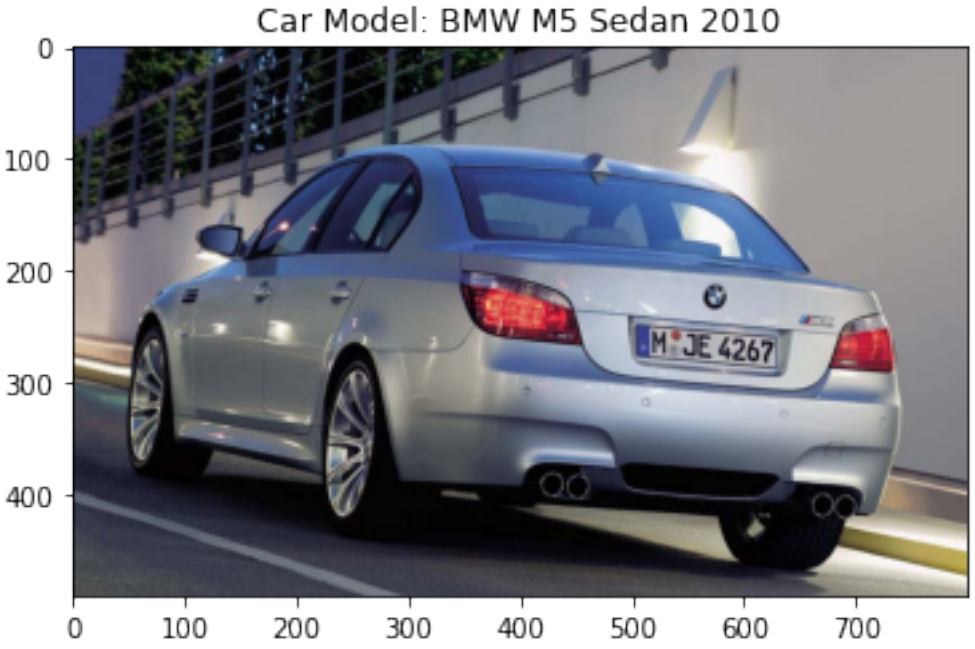
[ ]:
```
## loading the data
train_car = glob("/content/drive/My Drive/Colab Notebooks/car_data/train/*/*")
test_car = glob("/content/drive/My Drive/Colab Notebooks/car_data/test/*/*")
```

[ ]:
```
## getting the path to the data
train_path = "/content/drive/My Drive/Colab Notebooks/car_data/train/"
test_path = "/content/drive/My Drive/Colab Notebooks/car_data/test/"
```

[ ]:
```
def get_car_class(car):
  '''
  This function will return the car label/class per given image
  '''
    car_class = car.replace("/","").replace("\\","")[48::][:-9]
    return car_class
```

[ ]:
```
## showing some car images and their classes
for i in range(1,3500,700):
    image = Image.open(train_car[i])
    label = get_car_class(train_car[i])
    imgplot = plt.imshow(image)
    plt.title(f"Car Model: {label}")
    plt.show()
```


Car Model: Audi TT RS Coupe 2012
0
25
50
75
100
0
50
100
150
200
250


Car Model: BMW M5 Sedan 2010
0
100
200
300
400
0
100
200
300
400
500
600
700

[ ]:
```
import pandas as pd
## loading the labels/ car classes
```

```
car_models = pd.read_csv("/content/drive/My Drive/Colab Notebooks/car_data/
 ↪names.csv",
                                                                delimiter=';')
car_models.head()
```

[ ]:
```
                          Cars
0  AM General Hummer SUV 2000
1           Acura RL Sedan 2012
2           Acura TL Sedan 2012
3         Acura TL Type-S 2008
4         Acura TSX Sedan 2012
```

[ ]:
```
## creating a list with car classes
model_names = list(car_models['Cars'])
```

2.1 Image Augmentation

[ ]:
```
## setting up some parameters for data augmentation
img_width, img_height = 224, 224
train_samples = len(train_car)
validation_samples = len(test_car)
## there are 196 different models
n_classes = len(model_names)
batch_size = 32
```

[ ]:
```
from keras.preprocessing.image import ImageDataGenerator

## performing augmentation on the training data
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    zoom_range=0.2,
    rotation_range = 5,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)
```

[ ]:
```
## converting data to a tf.data.Dataset object
train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(img_width, img_height),
```

```
    batch_size=batch_size,
    class_mode='categorical')
```

```
Found 8144 images belonging to 196 classes.
Found 8041 images belonging to 196 classes.
```

3 Costumized CNN

I tried to customize a convolutional neural network (model) from scratch, I used with 3 Convolutional layers and two dense layers.

[ ]:
```
## Constructing a CNN with 3 Convolutional layers and two dense layers.
model = Sequential([
    Conv2D(16, 3, padding='same', activation='relu',
           input_shape=(img_width, img_height,3)),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(196, activation='softmax')
])

## checking the layers of the network
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 224, 224, 16)      448
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 112, 112, 32)      4640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               6422656
```

```
_________________________________________________________________
dense_3 (Dense)              (None, 196)               25284
=================================================================
Total params: 6,471,524
Trainable params: 6,471,524
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
## define optimizer
opt = tensorflow.keras.optimizers.Adam(lr=0.0001)

## compile model, define optimizer and the loss function
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

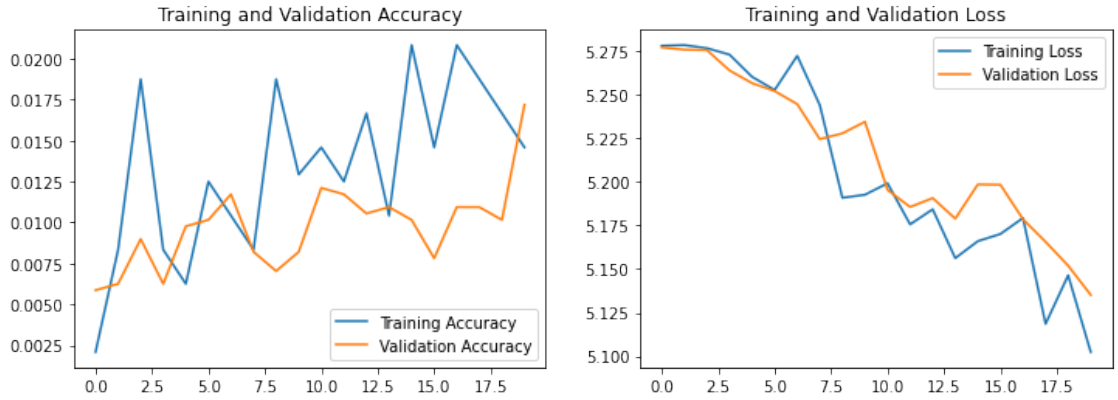
[ ]:
```
## train the model
history_0 = model.fit_generator(train_generator,
                            steps_per_epoch=15,
                            validation_data=validation_generator,
                           validation_steps=5, epochs=20)
```

```
Epoch 1/20
15/15 [==============================] - 73s 5s/step - loss: 5.2781 - accuracy:
0.0021 - val_loss: 5.2771 - val_accuracy: 0.0059
Epoch 2/20
15/15 [==============================] - 551s 37s/step - loss: 5.2785 -
accuracy: 0.0083 - val_loss: 5.2758 - val_accuracy: 0.0063
Epoch 3/20
15/15 [==============================] - 393s 26s/step - loss: 5.2767 -
accuracy: 0.0188 - val_loss: 5.2756 - val_accuracy: 0.0090
Epoch 4/20
15/15 [==============================] - 291s 19s/step - loss: 5.2730 -
accuracy: 0.0083 - val_loss: 5.2639 - val_accuracy: 0.0063
Epoch 5/20
15/15 [==============================] - 206s 14s/step - loss: 5.2602 -
accuracy: 0.0063 - val_loss: 5.2566 - val_accuracy: 0.0098
Epoch 6/20
15/15 [==============================] - 183s 12s/step - loss: 5.2527 -
accuracy: 0.0125 - val_loss: 5.2521 - val_accuracy: 0.0102
Epoch 7/20
15/15 [==============================] - 135s 9s/step - loss: 5.2723 - accuracy:
0.0104 - val_loss: 5.2446 - val_accuracy: 0.0117
Epoch 8/20
15/15 [==============================] - 109s 7s/step - loss: 5.2440 - accuracy:
0.0083 - val_loss: 5.2245 - val_accuracy: 0.0082
Epoch 9/20
15/15 [==============================] - 88s 6s/step - loss: 5.1909 - accuracy:
```

```
0.0188 - val_loss: 5.2278 - val_accuracy: 0.0070
Epoch 10/20
15/15 [==============================] - 73s 5s/step - loss: 5.1926 - accuracy:
0.0129 - val_loss: 5.2345 - val_accuracy: 0.0082
Epoch 11/20
15/15 [==============================] - 73s 5s/step - loss: 5.1992 - accuracy:
0.0146 - val_loss: 5.1954 - val_accuracy: 0.0121
Epoch 12/20
15/15 [==============================] - 68s 5s/step - loss: 5.1756 - accuracy:
0.0125 - val_loss: 5.1856 - val_accuracy: 0.0117
Epoch 13/20
15/15 [==============================] - 66s 4s/step - loss: 5.1843 - accuracy:
0.0167 - val_loss: 5.1907 - val_accuracy: 0.0105
Epoch 14/20
15/15 [==============================] - 58s 4s/step - loss: 5.1563 - accuracy:
0.0104 - val_loss: 5.1789 - val_accuracy: 0.0109
Epoch 15/20
15/15 [==============================] - 54s 4s/step - loss: 5.1660 - accuracy:
0.0208 - val_loss: 5.1985 - val_accuracy: 0.0102
Epoch 16/20
15/15 [==============================] - 49s 3s/step - loss: 5.1701 - accuracy:
0.0146 - val_loss: 5.1984 - val_accuracy: 0.0078
Epoch 17/20
15/15 [==============================] - 50s 3s/step - loss: 5.1793 - accuracy:
0.0208 - val_loss: 5.1786 - val_accuracy: 0.0109
Epoch 18/20
15/15 [==============================] - 51s 3s/step - loss: 5.1186 - accuracy:
0.0188 - val_loss: 5.1656 - val_accuracy: 0.0109
Epoch 19/20
15/15 [==============================] - 50s 3s/step - loss: 5.1464 - accuracy:
0.0167 - val_loss: 5.1520 - val_accuracy: 0.0102
Epoch 20/20
15/15 [==============================] - 46s 3s/step - loss: 5.1024 - accuracy:
0.0146 - val_loss: 5.1352 - val_accuracy: 0.0172
```

[ ]:
```
acc = history_0.history['accuracy']
val_acc = history_0.history['val_accuracy']

loss = history_0.history['loss']
val_loss = history_0.history['val_loss']

epochs_range = range(20)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
```

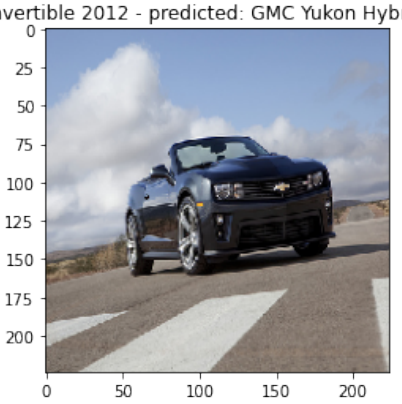
```
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



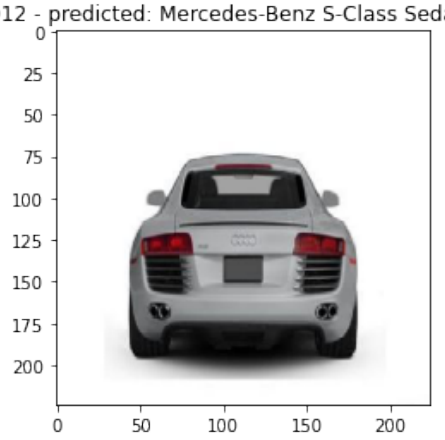
[ ]:
```
def predict_class(model):
    '''
    This function will predict what is the next car, check whether the␣
 ↪prediction
    was correct and lastly plot the image of the car
    '''
    image_batch, classes_batch = next(validation_generator)
    predicted_batch = model.predict(image_batch)
    for i in range(0,3):
        image = image_batch[i]
        pred = predicted_batch[i]
        the_pred = np.argmax(pred)
        predicted = model_names[the_pred]
        val_pred = max(pred)
        the_class = np.argmax(classes_batch[i])
        value = model_names[np.argmax(classes_batch[i])]
        plt.figure(i)
        isTrue = (the_pred == the_class)
        plt.title(str(isTrue) + ' class: ' + value + ' - ' +\
              'predicted: ' + predicted + ' with probability of: '\
               + str(val_pred)[:5])
        plt.imshow(image)
```

[ ]:
```
predict_class(model)
```

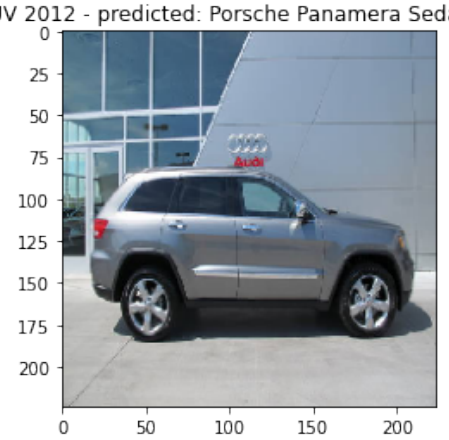
False class: Chevrolet Corvette Convertible 2012 - predicted: GMC Yukon Hybrid SUV 2012 with probability of: 0.009



False class: Audi R8 Coupe 2012 - predicted: Mercedes-Benz S-Class Sedan 2012 with probability of: 0.009



False class: Jeep Wrangler SUV 2012 - predicted: Porsche Panamera Sedan 2012 with probability of: 0.008



4 Transfer Learning

Transfer learning is a process where you take an existing trained model. In this project, we used three different models MobileNet V2, VGG16 and EfficientNet B1.

4.1 Mobile Net V2

MobileNet-v2 is a convolutional neural network that contains 53 layers. This network has been trained on more than a million images from the ImageNet database and it can classify objects into more than a 1000 categories. This CNN has been optimized to perform well on mobile devices.

[ ]:
```
## trasfer learning using mobile net
mobilenet_model = MobileNetV2(include_top=False,
                                    weights='imagenet',
                                    input_shape=(224, 224, 3))


## Change all layers to non-trainable
for layer in mobilenet_model.layers:
    layer.trainable = False

## adding some extra layers
x = GlobalAveragePooling2D()(mobilenet_model.output)
x = BatchNormalization()(x)
x = Dense(units=1024,
          activation='relu',kernel_regularizer=regularizers.l2(0.01),
          kernel_initializer='random_uniform',
          bias_initializer='zeros')(x)

x = BatchNormalization()(x)
output = Dense(units=196, activation='softmax')(x)

## creating the extended model
model_1 = Model(inputs=mobilenet_model.input, outputs=output)
```

[ ]:
```
model_1.summary()
```

```
Model: "functional_35"
_________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
=================================================================================
==================
input_25 (InputLayer)           [(None, 224, 224, 3) 0
_________________________________________________________________________________
__________________
Conv1_pad (ZeroPadding2D)       (None, 225, 225, 3)  0           input_25[0][0]
_________________________________________________________________________________
```

```
------------------
Conv1 (Conv2D)                  (None, 112, 112, 32) 864         Conv1_pad[0][0]
--------------------------------------------------------------------------------
------------------
bn_Conv1 (BatchNormalization)   (None, 112, 112, 32) 128         Conv1[0][0]
--------------------------------------------------------------------------------
------------------
Conv1_relu (ReLU)               (None, 112, 112, 32) 0           bn_Conv1[0][0]
--------------------------------------------------------------------------------
------------------
expanded_conv_depthwise (Depthw (None, 112, 112, 32) 288
Conv1_relu[0][0]
--------------------------------------------------------------------------------
------------------
expanded_conv_depthwise_BN (Bat (None, 112, 112, 32) 128
expanded_conv_depthwise[0][0]
--------------------------------------------------------------------------------
------------------
expanded_conv_depthwise_relu (R (None, 112, 112, 32) 0
expanded_conv_depthwise_BN[0][0]
--------------------------------------------------------------------------------
------------------
expanded_conv_project (Conv2D)  (None, 112, 112, 16) 512
expanded_conv_depthwise_relu[0][0
--------------------------------------------------------------------------------
------------------
expanded_conv_project_BN (Batch (None, 112, 112, 16) 64
expanded_conv_project[0][0]
--------------------------------------------------------------------------------
------------------
block_1_expand (Conv2D)         (None, 112, 112, 96) 1536
expanded_conv_project_BN[0][0]
--------------------------------------------------------------------------------
------------------
block_1_expand_BN (BatchNormali (None, 112, 112, 96) 384
block_1_expand[0][0]
--------------------------------------------------------------------------------
------------------
block_1_expand_relu (ReLU)      (None, 112, 112, 96) 0
block_1_expand_BN[0][0]
--------------------------------------------------------------------------------
------------------
block_1_pad (ZeroPadding2D)     (None, 113, 113, 96) 0
block_1_expand_relu[0][0]
--------------------------------------------------------------------------------
------------------
block_1_depthwise (DepthwiseCon (None, 56, 56, 96)   864
block_1_pad[0][0]
```

```
----------------------------------------------------------------------------------
------------------
block_1_depthwise_BN (BatchNorm (None, 56, 56, 96)   384
block_1_depthwise[0][0]
----------------------------------------------------------------------------------
------------------
block_1_depthwise_relu (ReLU)   (None, 56, 56, 96)   0
block_1_depthwise_BN[0][0]
----------------------------------------------------------------------------------
------------------
block_1_project (Conv2D)        (None, 56, 56, 24)   2304
block_1_depthwise_relu[0][0]
----------------------------------------------------------------------------------
------------------
block_1_project_BN (BatchNormal (None, 56, 56, 24)   96
block_1_project[0][0]
----------------------------------------------------------------------------------
------------------
block_2_expand (Conv2D)         (None, 56, 56, 144)  3456
block_1_project_BN[0][0]
----------------------------------------------------------------------------------
------------------
block_2_expand_BN (BatchNormali (None, 56, 56, 144)  576
block_2_expand[0][0]
----------------------------------------------------------------------------------
------------------
block_2_expand_relu (ReLU)      (None, 56, 56, 144)  0
block_2_expand_BN[0][0]
----------------------------------------------------------------------------------
------------------
block_2_depthwise (DepthwiseCon (None, 56, 56, 144)  1296
block_2_expand_relu[0][0]
----------------------------------------------------------------------------------
------------------
block_2_depthwise_BN (BatchNorm (None, 56, 56, 144)  576
block_2_depthwise[0][0]
----------------------------------------------------------------------------------
------------------
block_2_depthwise_relu (ReLU)   (None, 56, 56, 144)  0
block_2_depthwise_BN[0][0]
----------------------------------------------------------------------------------
------------------
block_2_project (Conv2D)        (None, 56, 56, 24)   3456
block_2_depthwise_relu[0][0]
----------------------------------------------------------------------------------
------------------
block_2_project_BN (BatchNormal (None, 56, 56, 24)   96
block_2_project[0][0]
```

```
------------------
dense_35 (Dense)                (None, 1024)         1311744
batch_normalization_30[0][0]

------------------------------------------------------------------------------
------------------
batch_normalization_31 (BatchNo (None, 1024)         4096        dense_35[0][0]

------------------------------------------------------------------------------
------------------
dense_36 (Dense)                (None, 196)          200900
batch_normalization_31[0][0]
==============================================================================
==================
Total params: 3,779,844
Trainable params: 1,517,252
Non-trainable params: 2,262,592

------------------------------------------------------------------------------
------------------
```

[ ]:
```
## compile the model, define optimizer and the loss function
opt = tensorflow.keras.optimizers.Adam(lr=0.0001)

model_1.compile(loss='categorical_crossentropy',
              optimizer=opt, metrics=['accuracy'])
```

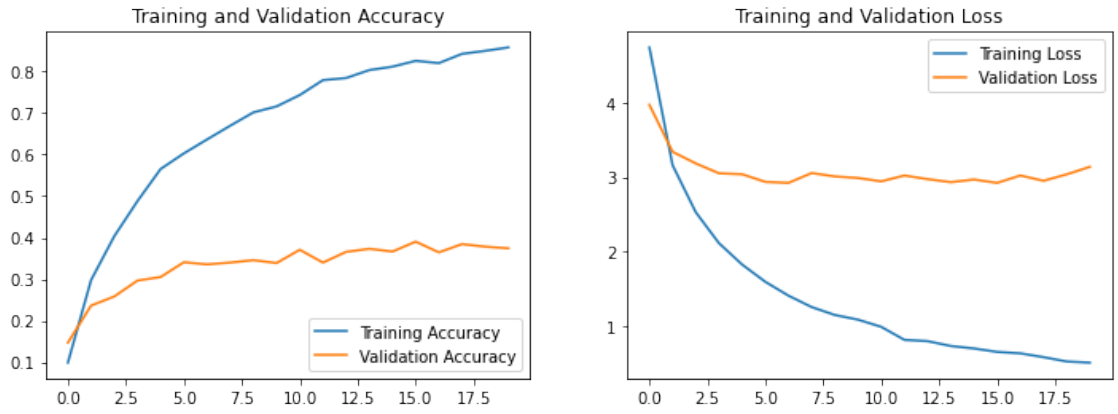
[ ]:
```
## train the model
history_1 = model_1.fit_generator(train_generator,
                           steps_per_epoch=len(train_generator),
                           validation_data=validation_generator,
                          validation_steps=len(validation_generator), epochs=20)
```

```
Epoch 1/20
255/255 [==============================] - 214s 840ms/step - loss: 15.2798 -
accuracy: 0.0592 - val_loss: 13.8995 - val_accuracy: 0.1210
Epoch 2/20
255/255 [==============================] - 211s 828ms/step - loss: 12.6481 -
accuracy: 0.2229 - val_loss: 12.1599 - val_accuracy: 0.2077
Epoch 3/20
255/255 [==============================] - 211s 827ms/step - loss: 10.8491 -
accuracy: 0.3777 - val_loss: 10.8359 - val_accuracy: 0.2599
Epoch 4/20
255/255 [==============================] - 211s 827ms/step - loss: 9.3964 -
accuracy: 0.4838 - val_loss: 9.7350 - val_accuracy: 0.3000
Epoch 5/20
255/255 [==============================] - 211s 827ms/step - loss: 8.1648 -
accuracy: 0.5861 - val_loss: 8.8061 - val_accuracy: 0.3258
Epoch 6/20
255/255 [==============================] - 212s 829ms/step - loss: 7.1260 -
accuracy: 0.6528 - val_loss: 8.0209 - val_accuracy: 0.3442
```

```
Epoch 7/20
255/255 [==============================] - 211s 827ms/step - loss: 6.2459 -
accuracy: 0.7167 - val_loss: 7.3440 - val_accuracy: 0.3608
Epoch 8/20
255/255 [==============================] - 215s 842ms/step - loss: 5.4676 -
accuracy: 0.7689 - val_loss: 6.7502 - val_accuracy: 0.3722
Epoch 9/20
255/255 [==============================] - 214s 840ms/step - loss: 4.8001 -
accuracy: 0.8078 - val_loss: 6.2590 - val_accuracy: 0.3748
Epoch 10/20
255/255 [==============================] - 210s 823ms/step - loss: 4.2117 -
accuracy: 0.8495 - val_loss: 5.8306 - val_accuracy: 0.3824
Epoch 11/20
255/255 [==============================] - 210s 824ms/step - loss: 3.7179 -
accuracy: 0.8757 - val_loss: 5.4366 - val_accuracy: 0.3900
Epoch 12/20
255/255 [==============================] - 211s 828ms/step - loss: 3.2930 -
accuracy: 0.8931 - val_loss: 5.1330 - val_accuracy: 0.3904
Epoch 13/20
255/255 [==============================] - 210s 824ms/step - loss: 2.9200 -
accuracy: 0.9111 - val_loss: 4.8275 - val_accuracy: 0.3931
Epoch 14/20
255/255 [==============================] - 210s 823ms/step - loss: 2.5941 -
accuracy: 0.9239 - val_loss: 4.5868 - val_accuracy: 0.3998
Epoch 15/20
255/255 [==============================] - 211s 826ms/step - loss: 2.3175 -
accuracy: 0.9334 - val_loss: 4.3831 - val_accuracy: 0.3967
Epoch 16/20
255/255 [==============================] - 210s 823ms/step - loss: 2.0827 -
accuracy: 0.9370 - val_loss: 4.2163 - val_accuracy: 0.4043
Epoch 17/20
255/255 [==============================] - 211s 827ms/step - loss: 1.8766 -
accuracy: 0.9436 - val_loss: 4.0558 - val_accuracy: 0.3995
Epoch 18/20
255/255 [==============================] - 211s 828ms/step - loss: 1.6887 -
accuracy: 0.9499 - val_loss: 3.9101 - val_accuracy: 0.4032
Epoch 19/20
255/255 [==============================] - 210s 824ms/step - loss: 1.5370 -
accuracy: 0.9514 - val_loss: 3.7910 - val_accuracy: 0.3998
Epoch 20/20
255/255 [==============================] - 211s 826ms/step - loss: 1.4134 -
accuracy: 0.9506 - val_loss: 3.7134 - val_accuracy: 0.4044
```

[ ]:
```
acc_1 = history_1.history['accuracy']
val_acc_1 = history_1.history['val_accuracy']

loss_1 = history_1.history['loss']
```

```
val_loss_1 = history_1.history['val_loss']

epochs_range = range(20)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc_1, label='Training Accuracy')
plt.plot(epochs_range, val_acc_1, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss_1, label='Training Loss')
plt.plot(epochs_range, val_loss_1, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



[ ]:
```
predict_class(model_1)
```

4.2 VGG 16

Check here to see VGG16 implementation (kernel died and it was taking too much time to rerun everything)

4.3 EfficientNet B1

EfficientNet is among the most efficient models Also, this network has been pre-trained on the ImageNet database, but it is considered to be one the most efficient models. In comparsion with other CNNs, EfficientNet use a method called compound scaling, which uniformly scales all dimensions of depth/width/resolution while maintaingn the balance (ref). That reduces significantly the the dimensions of the classification problems.

[ ]:
```
from keras.layers import GlobalAveragePooling2D, Dense, BatchNormalization
from keras import Model, optimizers

## loading the EfficientNetB1 model
base_model = efn.EfficientNetB1(weights='imagenet', include_top=False)

## adding some extra layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(196, activation='softmax')(x)
model_2 = Model(inputs=base_model.input, outputs=predictions)

## fix the feature extraction part of the model
for layer in base_model.layers:
    if isinstance(layer, BatchNormalization):
        layer.trainable = True
    else:
        layer.trainable = False

model_2.summary()
```

```
Downloading data from https://github.com/Callidior/keras-applications/releases/d
ownload/efficientnet/efficientnet-b1_weights_tf_dim_ordering_tf_kernels_autoaugm
ent_notop.h5
27164672/27164032 [==============================] - 0s 0us/step
Model: "functional_1"
__________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
input_1 (InputLayer)            [(None, None, None,  0
__________________________________________________________________________________
__________________
stem_conv (Conv2D)              (None, None, None, 3 864         input_1[0][0]
__________________________________________________________________________________
__________________
stem_bn (BatchNormalization)    (None, None, None, 3 128         stem_conv[0][0]
__________________________________________________________________________________
__________________
stem_activation (Activation)    (None, None, None, 3 0           stem_bn[0][0]
__________________________________________________________________________________
__________________
block1a_dwconv (DepthwiseConv2D (None, None, None, 3 288
stem_activation[0][0]
__________________________________________________________________________________
__________________
```

```
block1a_bn (BatchNormalization) (None, None, None, 3 128
block1a_dwconv[0][0]
__________________________________________________________________________________
__________________
block1a_activation (Activation) (None, None, None, 3 0
block1a_bn[0][0]
__________________________________________________________________________________
__________________
block1a_se_squeeze (GlobalAvera (None, 32)           0
block1a_activation[0][0]
__________________________________________________________________________________
__________________
block1a_se_reshape (Reshape)    (None, 1, 1, 32)     0
block1a_se_squeeze[0][0]
__________________________________________________________________________________
__________________
block1a_se_reduce (Conv2D)      (None, 1, 1, 8)      264
block1a_se_reshape[0][0]
__________________________________________________________________________________
__________________
block1a_se_expand (Conv2D)      (None, 1, 1, 32)     288
block1a_se_reduce[0][0]
__________________________________________________________________________________
__________________
block1a_se_excite (Multiply)    (None, None, None, 3 0
block1a_activation[0][0]
block1a_se_expand[0][0]
__________________________________________________________________________________
__________________
block1a_project_conv (Conv2D)   (None, None, None, 1 512
block1a_se_excite[0][0]
__________________________________________________________________________________
__________________
block1a_project_bn (BatchNormal (None, None, None, 1 64
block1a_project_conv[0][0]
__________________________________________________________________________________
__________________
block1b_dwconv (DepthwiseConv2D (None, None, None, 1 144
block1a_project_bn[0][0]
__________________________________________________________________________________
__________________
block1b_bn (BatchNormalization) (None, None, None, 1 64
block1b_dwconv[0][0]
__________________________________________________________________________________
__________________
block1b_activation (Activation) (None, None, None, 1 0
block1b_bn[0][0]
__________________________________________________________________________________
```

```
------------------
block1b_se_squeeze (GlobalAvera (None, 16)           0
block1b_activation[0][0]
------------------------------------------------------------------------------
------------------
block1b_se_reshape (Reshape)    (None, 1, 1, 16)     0
block1b_se_squeeze[0][0]
------------------------------------------------------------------------------
------------------
block1b_se_reduce (Conv2D)      (None, 1, 1, 4)      68
block1b_se_reshape[0][0]
------------------------------------------------------------------------------
------------------
block1b_se_expand (Conv2D)      (None, 1, 1, 16)     80
block1b_se_reduce[0][0]
------------------------------------------------------------------------------
------------------
block1b_se_excite (Multiply)    (None, None, None, 1 0
block1b_activation[0][0]
block1b_se_expand[0][0]
------------------------------------------------------------------------------
------------------
block1b_project_conv (Conv2D)   (None, None, None, 1 256
block1b_se_excite[0][0]
------------------------------------------------------------------------------
------------------
block1b_project_bn (BatchNormal (None, None, None, 1 64
block1b_project_conv[0][0]
------------------------------------------------------------------------------
------------------
block1b_drop (FixedDropout)     (None, None, None, 1 0
block1b_project_bn[0][0]
------------------------------------------------------------------------------
------------------
block1b_add (Add)               (None, None, None, 1 0
block1b_drop[0][0]
block1a_project_bn[0][0]
------------------------------------------------------------------------------
------------------
block2a_expand_conv (Conv2D)    (None, None, None, 9 1536
block1b_add[0][0]
------------------------------------------------------------------------------
------------------
block2a_expand_bn (BatchNormali (None, None, None, 9 384
block2a_expand_conv[0][0]
------------------------------------------------------------------------------
------------------
block2a_expand_activation (Acti (None, None, None, 9 0
```

```
------------------
block7b_se_squeeze (GlobalAvera (None, 1920)         0
block7b_activation[0][0]
------------------------------------------------------------------------------
------------------
block7b_se_reshape (Reshape)    (None, 1, 1, 1920)   0
block7b_se_squeeze[0][0]
------------------------------------------------------------------------------
------------------
block7b_se_reduce (Conv2D)      (None, 1, 1, 80)     153680
block7b_se_reshape[0][0]
------------------------------------------------------------------------------
------------------
block7b_se_expand (Conv2D)      (None, 1, 1, 1920)   155520
block7b_se_reduce[0][0]
------------------------------------------------------------------------------
------------------
block7b_se_excite (Multiply)    (None, None, None, 1 0
block7b_activation[0][0]
block7b_se_expand[0][0]
------------------------------------------------------------------------------
------------------
block7b_project_conv (Conv2D)   (None, None, None, 3 614400
block7b_se_excite[0][0]
------------------------------------------------------------------------------
------------------
block7b_project_bn (BatchNormal (None, None, None, 3 1280
block7b_project_conv[0][0]
------------------------------------------------------------------------------
------------------
block7b_drop (FixedDropout)     (None, None, None, 3 0
block7b_project_bn[0][0]
------------------------------------------------------------------------------
------------------
block7b_add (Add)               (None, None, None, 3 0
block7b_drop[0][0]
block7a_project_bn[0][0]
------------------------------------------------------------------------------
------------------
top_conv (Conv2D)               (None, None, None, 1 409600
block7b_add[0][0]
------------------------------------------------------------------------------
------------------
top_bn (BatchNormalization)     (None, None, None, 1 5120        top_conv[0][0]
------------------------------------------------------------------------------
------------------
top_activation (Activation)     (None, None, None, 1 0           top_bn[0][0]
------------------------------------------------------------------------------
```

```
------------------
global_average_pooling2d (Globa (None, 1280)         0
top_activation[0][0]
__________________________________________________________________________________
------------------
dense_4 (Dense)                 (None, 196)          251076
global_average_pooling2d[0][0]
==================================================================================
==================
Total params: 6,826,308
Trainable params: 313,124
Non-trainable params: 6,513,184
__________________________________________________________________________________
------------------
```

[ ]:
```
## compile model, define optimizer and the loss function
model_2.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(lr=0.01), metrics=['accuracy'])
```

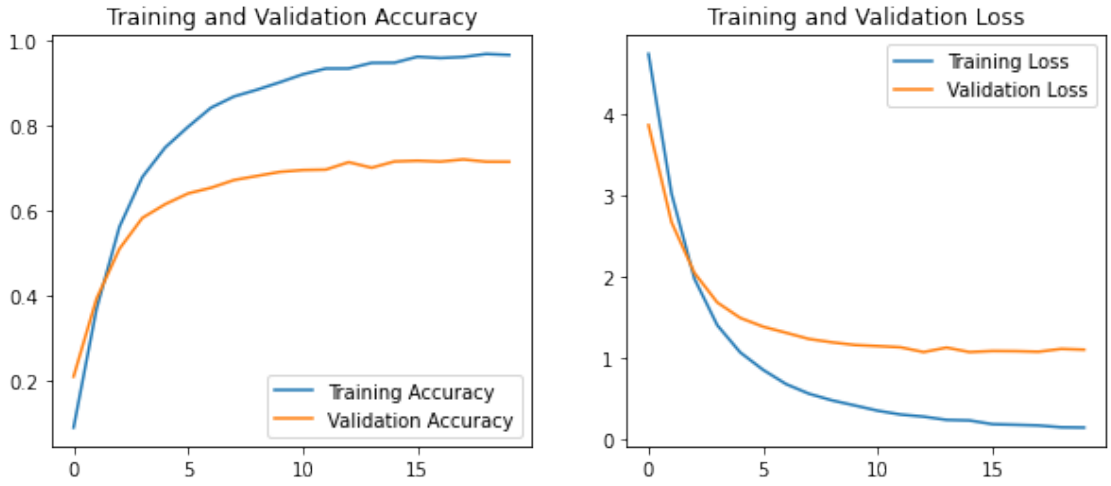
[ ]:
```
## train the model
history_2 = model_2.fit_generator(generator=train_generator,
                    steps_per_epoch=len(train_generator) ,
                    validation_data=validation_generator,
                    validation_steps=len(validation_generator),
                    epochs=20)
```

```
Epoch 1/20
255/255 [==============================] - 206s 807ms/step - loss: 4.7295 -
accuracy: 0.0905 - val_loss: 3.8548 - val_accuracy: 0.2104
Epoch 2/20
255/255 [==============================] - 202s 790ms/step - loss: 3.0157 -
accuracy: 0.3692 - val_loss: 2.6637 - val_accuracy: 0.3934
Epoch 3/20
255/255 [==============================] - 201s 789ms/step - loss: 1.9686 -
accuracy: 0.5624 - val_loss: 2.0400 - val_accuracy: 0.5106
Epoch 4/20
255/255 [==============================] - 201s 787ms/step - loss: 1.3963 -
accuracy: 0.6795 - val_loss: 1.6762 - val_accuracy: 0.5834
Epoch 5/20
255/255 [==============================] - 202s 791ms/step - loss: 1.0631 -
accuracy: 0.7491 - val_loss: 1.4870 - val_accuracy: 0.6157
Epoch 6/20
255/255 [==============================] - 201s 788ms/step - loss: 0.8484 -
accuracy: 0.7978 - val_loss: 1.3790 - val_accuracy: 0.6410
Epoch 7/20
255/255 [==============================] - 201s 786ms/step - loss: 0.6741 -
accuracy: 0.8426 - val_loss: 1.3045 - val_accuracy: 0.6543
Epoch 8/20
```

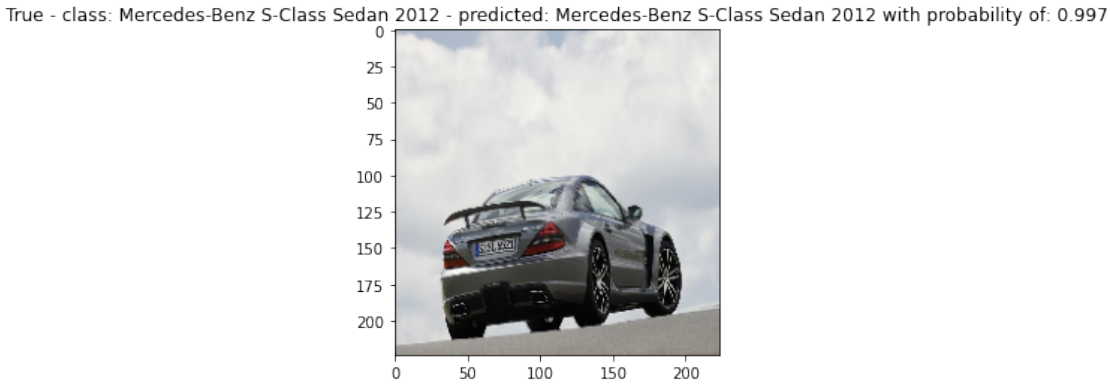
```
255/255 [==============================] - 201s 787ms/step - loss: 0.5553 -
accuracy: 0.8689 - val_loss: 1.2279 - val_accuracy: 0.6724
Epoch 9/20
255/255 [==============================] - 201s 786ms/step - loss: 0.4743 -
accuracy: 0.8847 - val_loss: 1.1869 - val_accuracy: 0.6820
Epoch 10/20
255/255 [==============================] - 200s 783ms/step - loss: 0.4112 -
accuracy: 0.9023 - val_loss: 1.1542 - val_accuracy: 0.6916
Epoch 11/20
255/255 [==============================] - 201s 786ms/step - loss: 0.3467 -
accuracy: 0.9209 - val_loss: 1.1407 - val_accuracy: 0.6958
Epoch 12/20
255/255 [==============================] - 199s 782ms/step - loss: 0.2999 -
accuracy: 0.9344 - val_loss: 1.1261 - val_accuracy: 0.6968
Epoch 13/20
255/255 [==============================] - 201s 788ms/step - loss: 0.2742 -
accuracy: 0.9346 - val_loss: 1.0672 - val_accuracy: 0.7143
Epoch 14/20
255/255 [==============================] - 201s 788ms/step - loss: 0.2343 -
accuracy: 0.9478 - val_loss: 1.1227 - val_accuracy: 0.7013
Epoch 15/20
255/255 [==============================] - 201s 789ms/step - loss: 0.2276 -
accuracy: 0.9482 - val_loss: 1.0673 - val_accuracy: 0.7161
Epoch 16/20
255/255 [==============================] - 200s 786ms/step - loss: 0.1815 -
accuracy: 0.9623 - val_loss: 1.0808 - val_accuracy: 0.7178
Epoch 17/20
255/255 [==============================] - 201s 786ms/step - loss: 0.1743 -
accuracy: 0.9592 - val_loss: 1.0789 - val_accuracy: 0.7157
Epoch 18/20
255/255 [==============================] - 201s 789ms/step - loss: 0.1659 -
accuracy: 0.9617 - val_loss: 1.0714 - val_accuracy: 0.7209
Epoch 19/20
255/255 [==============================] - 201s 790ms/step - loss: 0.1432 -
accuracy: 0.9688 - val_loss: 1.1065 - val_accuracy: 0.7155
Epoch 20/20
255/255 [==============================] - 202s 792ms/step - loss: 0.1398 -
accuracy: 0.9665 - val_loss: 1.0970 - val_accuracy: 0.7153
```

[ ]:
```
acc_2 = history_2.history['accuracy']
val_acc_2 = history_2.history['val_accuracy']

loss_2 = history_2.history['loss']
val_loss_2 = history_2.history['val_loss']

epochs_range = range(20)
```

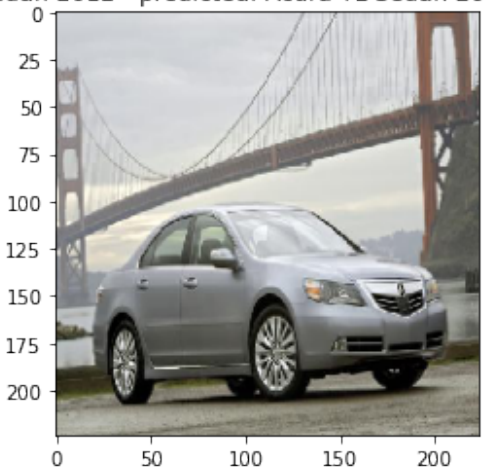
```
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc_2, label='Training Accuracy')
plt.plot(epochs_range, val_acc_2, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss_2, label='Training Loss')
plt.plot(epochs_range, val_loss_2, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



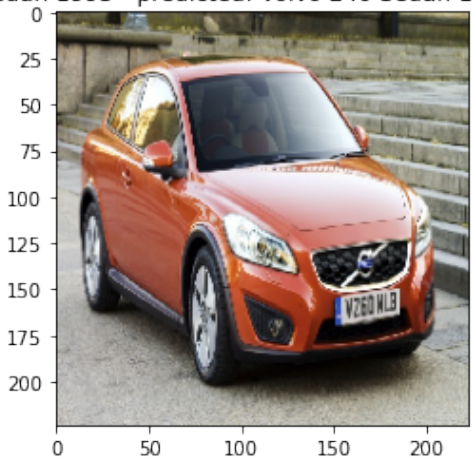
[ ]:
```
## making prediction about different car models
predict_class(model_2)
```



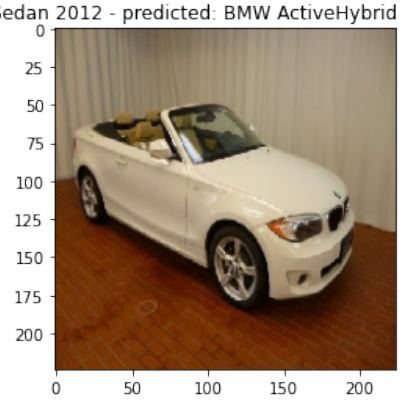
True - class: Acura TL Sedan 2012 - predicted: Acura TL Sedan 2012 with probability of: 0.759



True - class: Volvo 240 Sedan 1993 - predicted: Volvo 240 Sedan 1993 with probability of: 0.996

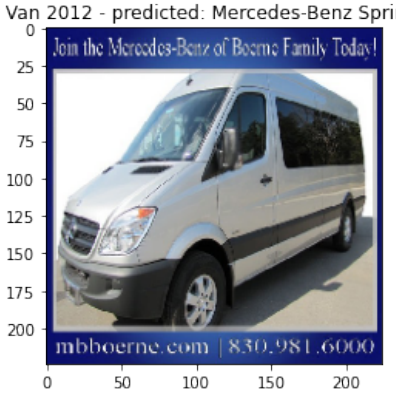


True - class: BMW ActiveHybrid 5 Sedan 2012 - predicted: BMW ActiveHybrid 5 Sedan 2012 with probability of: 0.967

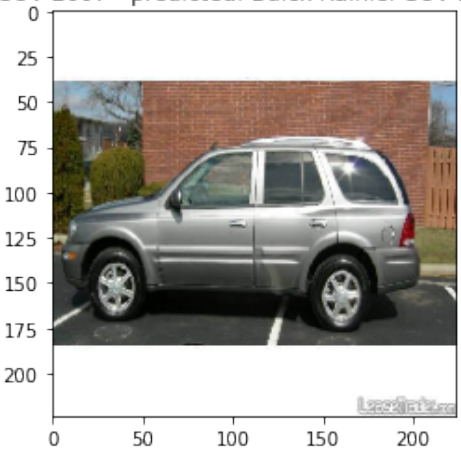


False - class: Toyota Corolla Sedan 2012 - predicted: Mitsubishi Lancer Sedan 2012 with probability of: 0.317

True - class: Mercedes-Benz Sprinter Van 2012 - predicted: Mercedes-Benz Sprinter Van 2012 with probability of: 0.999



True - class: Buick Rainier SUV 2007 - predicted: Buick Rainier SUV 2007 with probability of: 0.638



Investigating other metrics

NOTE: to get the accurate results you have to set up the size of valid generator to 8144 (because of time constrains I won't be able to rerun it again)

[ ]:
```
from sklearn.metrics import classification_report, confusion_matrix

## evaluating the model
evaluation = model_2.evaluate_generator(validation_generator)
print("Testing accuracy = ",evaluation[1])

## printing the confusion matrix
for i,j in validation_generator:
    print(i.shape, j.shape)
    p = model.predict(i)
    p = p.argmax(-1)
    t = j.argmax(-1)
    print(classification_report(t,p))
    #print(confusion_matrix(t,p))
    break;
```

```
Testing accuracy =  0.7153339385986328
(150, 224, 224, 3) (150, 196)
              precision    recall  f1-score   support

           0       0.33      0.50      0.40         2
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         1
           4       0.00      0.00      0.00         3
           5       0.00      0.00      0.00         2
           6       0.00      0.00      0.00         3
           7       0.00      0.00      0.00         2
           8       0.00      0.00      0.00         1
           9       0.00      0.00      0.00         1
          11       0.00      0.00      0.00         1
          16       0.00      0.00      0.00         1
          19       0.00      0.00      0.00         1
          20       0.20      0.50      0.29         2
          21       0.00      0.00      0.00         2
          22       0.00      0.00      0.00         0
          26       0.00      0.00      0.00         1
          27       0.00      0.00      0.00         2
          28       0.00      0.00      0.00         1
          33       0.00      0.00      0.00         2
          36       0.00      0.00      0.00         1
          37       0.00      0.00      0.00         1
          40       0.00      0.00      0.00         1
          41       0.00      0.00      0.00         2
```

43	0.00	0.00	0.00	1
45	0.00	0.00	0.00	1
47	0.00	0.00	0.00	2
48	0.00	0.00	0.00	1
50	0.00	0.00	0.00	1
51	0.00	0.00	0.00	3
52	0.00	0.00	0.00	0
53	0.00	0.00	0.00	1
54	0.00	0.00	0.00	1
55	0.00	0.00	0.00	1
56	0.00	0.00	0.00	0
57	0.00	0.00	0.00	1
61	0.00	0.00	0.00	2
62	0.00	0.00	0.00	2
63	0.00	0.00	0.00	1
64	0.00	0.00	0.00	1
65	0.00	0.00	0.00	2
69	0.00	0.00	0.00	1
70	0.00	0.00	0.00	2
73	0.00	0.00	0.00	1
74	0.00	0.00	0.00	2
76	0.00	0.00	0.00	1
77	0.00	0.00	0.00	1
80	0.00	0.00	0.00	1
81	0.00	0.00	0.00	2
82	0.00	0.00	0.00	1
84	0.00	0.00	0.00	0
85	0.00	0.00	0.00	1
86	0.00	0.00	0.00	1
87	0.50	0.25	0.33	4
89	0.00	0.00	0.00	1
90	0.00	0.00	0.00	2
92	0.00	0.00	0.00	1
95	0.00	0.00	0.00	2
96	0.00	0.00	0.00	0
97	0.00	0.00	0.00	1
98	0.00	0.00	0.00	1
99	0.00	0.00	0.00	1
101	0.00	0.00	0.00	1
102	0.00	0.00	0.00	0
103	0.00	0.00	0.00	1
104	0.00	0.00	0.00	1
105	0.00	0.00	0.00	1
108	0.00	0.00	0.00	1
109	0.00	0.00	0.00	1
110	0.00	0.00	0.00	1
111	0.00	0.00	0.00	2
112	0.00	0.00	0.00	1

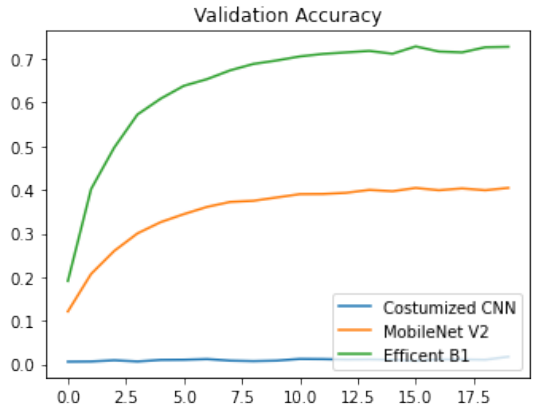
113	0.00	0.00	0.00	4
115	0.00	0.00	0.00	1
116	0.00	0.00	0.00	1
117	0.00	0.00	0.00	2
119	0.06	0.50	0.10	2
122	0.00	0.00	0.00	2
124	0.00	0.00	0.00	1
126	0.00	0.00	0.00	2
127	0.00	0.00	0.00	0
128	0.00	0.00	0.00	1
129	0.00	0.00	0.00	1
133	0.00	0.00	0.00	1
135	0.00	0.00	0.00	2
139	0.00	0.00	0.00	1
142	0.00	0.00	0.00	0
143	0.00	0.00	0.00	2
144	0.00	0.00	0.00	1
145	0.00	0.00	0.00	1
149	0.00	0.00	0.00	0
151	0.00	0.00	0.00	0
152	0.00	0.00	0.00	1
153	0.00	0.00	0.00	3
154	0.00	0.00	0.00	2
156	0.00	0.00	0.00	1
157	0.00	0.00	0.00	1
159	0.00	0.00	0.00	3
160	0.00	0.00	0.00	2
161	0.00	0.00	0.00	1
163	0.00	0.00	0.00	1
164	0.00	0.00	0.00	0
165	0.00	0.00	0.00	0
166	0.00	0.00	0.00	0
167	0.00	0.00	0.00	1
168	0.00	0.00	0.00	1
169	0.00	0.00	0.00	1
170	0.00	0.00	0.00	1
172	0.00	0.00	0.00	1
175	0.00	0.00	0.00	1
176	0.00	0.00	0.00	1
177	0.00	0.00	0.00	2
178	0.00	0.00	0.00	0
179	0.00	0.00	0.00	1
180	0.00	0.00	0.00	1
182	0.00	0.00	0.00	1
184	0.00	0.00	0.00	1
185	0.00	0.00	0.00	1
188	0.00	0.00	0.00	1
189	0.00	0.00	0.00	1

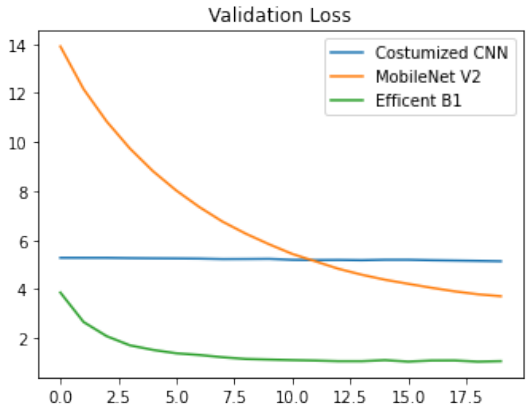
```
          191       0.00      0.00      0.00         1
          193       0.00      0.00      0.00         1
          194       0.00      0.00      0.00         0
          195       0.00      0.00      0.00         1

     accuracy                           0.03       150
    macro avg       0.01      0.01      0.01       150
 weighted avg       0.02      0.03      0.02       150


/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

5 Comparsion of the networks above

[ ]:
```
acc = history_0.history['accuracy']
val_acc = history_0.history['val_accuracy']

loss = history_0.history['loss']
val_loss = history_0.history['val_loss']

epochs_range = range(20)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)

plt.plot(epochs_range, val_acc, label='Costumized CNN')
plt.plot(epochs_range, val_acc_1, label='MobileNet V2')
plt.plot(epochs_range, val_acc_2, label='Efficent B1')
plt.legend(loc='lower right')
plt.title('Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, val_loss, label='Costumized CNN')
plt.plot(epochs_range, val_loss_1, label='MobileNet V2')
plt.plot(epochs_range, val_loss_2, label='Efficent B1')

plt.legend(loc='upper right')
plt.title('Validation Loss')
```

```
plt.show()
```


Validation Accuracy
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Costumized CNN
MobileNet V2
Efficent B1


Validation Loss
14
12
10
8
6
4
2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Costumized CNN
MobileNet V2
Efficent B1