ESGI
école supérieure de génie informatique

Report

Annual project

Creation of a deep learning library and an application with basic models

Car recognition app

2019 -2020

Group 11
Damien Smagghe

3A IABD1

Car brand classification report

Summary

1. Introduction

This report aims to explain how te evaluate various model for a same problematic : **Car brand prediction with an image dataset**. We'll be exploring various models and training various one te interpret which one is the most accurate. Then we'll explain intuition that led us to several way to imrove data and therefore the **model**.

This notebook contains three four part:

- Data analysis
- Data preprocessing
- Neural networks

2. Data analysis

This part will describe how the data was scrapped and generated and also will be explored to know the repartition of each car brand in the dataset.

2.1 Exploring the subject

First approach First of all, in order to retrieve numerous images of a car from a specific brand we decide to create a bot to scrap google images with search such as 'audi front' or 'mercedes'.

The results were not good with too many unrevelant photos and bad quality:

Entrée [1]:
```
#
# Firstly we import necessary libs
#
import selenium
import os
import io
import time
import requests
import hashlib
from PIL import Image
from selenium import webdriver
DRIVER_PATH = '../../chromedriver' # might need a different version of chromedriver depending on OS or chrome version
```

```
#
# Fetch all images url's from a specific gogole search
#

def fetch_image_urls(query:str, max_links_to_fetch:int, wd:webdriver, sleep_between_interactions:int=1):
    def scroll_to_end(wd):
        wd.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        time.sleep(sleep_between_interactions)

    # build the google query
    search_url = "https://www.google.com/search?safe=off&site=&tbm=isch&source=hp&q={q}&oq={q}&gs_l=img"

    # load the page
    wd.get(search_url.format(q=query))

    image_urls = set()
    image_count = 0
    results_start = 0
    while image_count < max_links_to_fetch:
        scroll_to_end(wd)

        # get all image thumbnail results
        thumbnail_results = wd.find_elements_by_css_selector("img.Q4LuWd")
        number_results = len(thumbnail_results)

        print(f"Found: {number_results} search results. Extracting links from {results_start}:{number_results}")

        for img in thumbnail_results[results_start:number_results]:
            # try to click every thumbnail such that we can get the real image behind it
            try:
                img.click()
                time.sleep(sleep_between_interactions)
            except Exception:
                continue

            # extract image urls
            actual_images = wd.find_elements_by_css_selector('img.n3VNCb')
            for actual_image in actual_images:
                if actual_image.get_attribute('src') and 'http' in actual_image.get_attribute('src'):
                    image_urls.add(actual_image.get_attribute('src'))

            image_count = len(image_urls)

            if len(image_urls) >= max_links_to_fetch:
                print(f"Found: {len(image_urls)} image links, done!")
                break
        else:
            print("Found:", len(image_urls), "image links, looking for more ...")
            time.sleep(30)
            return
            load_more_button = wd.find_element_by_css_selector(".mye4qd")
            if load_more_button:
                wd.execute_script("document.querySelector('.mye4qd').click();")

        # move the result startpoint further down
        results_start = len(thumbnail_results)

    return image_urls

#
# Utility function to save images form url in a folder
#

def persist_image(folder_path:str,url:str):
    try:
        image_content = requests.get(url).content

    except Exception as e:
        print(f"ERROR - Could not download {url} - {e}")

    try:
        image_file = io.BytesIO(image_content)
        image = Image.open(image_file).convert('RGB')
        file_path = os.path.join(folder_path,hashlib.sha1(image_content).hexdigest()[:10] + '.jpg')
        with open(file_path, 'wb') as f:
            image.save(f, "JPEG", quality=85)
        print(f"SUCCESS - saved {url} - as {file_path}")
    except Exception as e:
        print(f"ERROR - Could not save {url} - {e}")

#
# Combine fetch url and save image utility functions into a more practical to use function
#

def search_and_download(search_term:str,driver_path:str,target_path='./images',number_images=5):
    target_folder = os.path.join(target_path,'_'.join(search_term.lower().split(' ')))

    if not os.path.exists(target_folder):
        os.makedirs(target_folder)

    with webdriver.Chrome(executable_path=driver_path) as wd:
        res = fetch_image_urls(search_term, number_images, wd=wd, sleep_between_interactions=0.5)

    for elem in res:
        persist_image(target_folder,elem)

# Example of usage => search_and_download('audi front', DRIVER_PATH, './dataset', 60)
```

2.2 Generating the dataset

Second approach

From then on we wanted to base ourselves on the images of the car ad site. By dint of searching we found the site carizy.com which offered a strong formalised layout for pictures. By studying the site we found that the photos were hosted on a CDN with a formalised structure for each angle (front hood). So we set up a script to retrieve quality images on the brands we wanted with the right angle. By doing so we could iterate over brands and also over angles of the car.

Entrée [2]:

```
#
# Import necessary dependencies
#
import time
import requests
import os
import io
from PIL import Image #pip install Pillow
import hashlib

#
# Define a car data class that retrieve the useful data
#

class CarData:
    def __init__(self, car_id, url, img):
        self.car_id = car_id
        self.url = url
        self.img = img

    def __repr__(self):
        return str(self.car_id) + " : " + self.url + ' + ' + self.img

#
# Request the search api of the website with good params
#

def send_algolia_request(page):
    url = 'https://65quub6n61-dsn.algolia.net/1/indexes/*/queries?x-algolia-agent=Algolia%20for%20vanilla%20JavaScript
    data = {'requests': [{'indexName' : 'CarAlgoliaIndex_prod', 'params': 'query=&hitsPerPage=21&maxValuesPerFacet=100
    p = requests.post(url, json=data)
    return p.json()

#
# map data to keep only useful data
#

def map_data(_car_data, hits):
    for e in hits:
        if e['make'] in _car_data:
            _car_data[e['make']].append(CarData(e['carId'], e['lien'], e['photo']))
        else:
            _car_data[e['make']] = [CarData(e['carId'], e['lien'], e['photo'])]

#
# Make all the request with a interval to avoid api blocking
#

car_data = {}
i = 0
while i < 48:
    x = send_algolia_request(i)
    map_data(car_data, x['results'][0]['hits'])
    i += 1
    time.sleep(1)



#
# Utility function to download the images
#

def persist_image(folder_path:str,url:str):
    try:
        image_content = requests.get(url).content

    except Exception as e:
        print(f"ERROR - Could not download {url} - {e}")

    try:
        image_file = io.BytesIO(image_content)
        image = Image.open(image_file).convert('RGB')
        file_path = os.path.join(folder_path,hashlib.sha1(image_content).hexdigest()[:10] + '.jpg')
```

```
        with open(file_path, 'wb') as f:
            image.save(f, "JPEG", quality=85)
        print(f"SUCCESS - saved {url} - as {file_path}")
    except Exception as e:
        print(f"ERROR - Could not save {url} - {e}")

#
# Final function that takes brand and angle paramaters
#

def download_brand_angle_img(brand, angle, folder):
    urls = []
    folder_url = folder
    for x in car_data[brand]:

        ur = x.img.replace('avant-gauche', angle)
        finalurl = 'https://cdn.carizy.com/carphotos/' + str(x.car_id) + '/wide/' + ur
        urls.append(finalurl)

    for z in urls:
        persist_image(folder_url, z)
        time.sleep(0.2)
# Folder must be already created
#Example of usage => download_brand_angle_img('AUDI', 'face-capot', './cars')
```

We will know download raw data for three brands in the folder raw of the project structure.

Entrée [6]:
```
download_brand_angle_img('PEUGEOT', 'face-capot', '../../Dataset/Raw/peugeot')
```

```
SUCCESS - saved https://cdn.carizy.com/carphotos/34025/wide/peugeot-308-sw-occasion-2016-face-capot.jpg - as ../../Da
taset/Raw/peugeot/7fb59e79af.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34008/wide/peugeot-308-sw-occasion-2015-face-capot.jpg - as ../../Da
taset/Raw/peugeot/998d3538a9.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33994/wide/peugeot-108-occasion-2016-face-capot.jpg - as ../../Datas
et/Raw/peugeot/b4e245828e.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33903/wide/peugeot-308-cc-occasion-2013-face-capot.jpg - as ../../Da
taset/Raw/peugeot/d086289ac9.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33889/wide/peugeot-108-occasion-2014-face-capot.jpg - as ../../Datas
et/Raw/peugeot/4b6f0d71bb.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33878/wide/peugeot-308-sw-occasion-2014-face-capot.jpg - as ../../Da
taset/Raw/peugeot/a2148ee367.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33981/wide/peugeot-208-occasion-2018-face-capot.jpg - as ../../Datas
et/Raw/peugeot/fafc0070dc.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33968/wide/peugeot-207-cc-occasion-2010-face-capot.jpg - as ../../Da
taset/Raw/peugeot/bdd4063ac3.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33970/wide/peugeot-208-occasion-2015-face-capot.jpg - as ../../Datas
et/Raw/peugeot/152b9e48b8.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33952/wide/peugeot-3008-occasion-2012-face-capot.jpg - as ../../Data
set/Raw/peugeot/3a89b57cf0.jpg
```

Entrée [7]:
```
download_brand_angle_img('RENAULT', 'face-capot', '../../Dataset/Raw/renault')
```

```
SUCCESS - saved https://cdn.carizy.com/carphotos/34170/wide/renault-twingo-iii-occasion-2014-face-capot.jpg - as
../../Dataset/Raw/renault/d6d4ce5cf6.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34165/wide/renault-twingo-ii-occasion-2012-face-capot.jpg - as
../../Dataset/Raw/renault/319aff485d.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34109/wide/renault-twingo-ii-occasion-2010-face-capot.jpg - as
../../Dataset/Raw/renault/5990986b51.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34158/wide/renault-captur-occasion-2017-face-capot.jpg - as ../../Da
taset/Raw/renault/f065264b39.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34157/wide/renault-captur-occasion-2013-face-capot.jpg - as ../../Da
taset/Raw/renault/ffafed7e16.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34141/wide/renault-twingo-iii-occasion-2016-face-capot.jpg - as
../../Dataset/Raw/renault/30f8dd49c4.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34107/wide/renault-twingo-ii-occasion-2011-face-capot.jpg - as
../../Dataset/Raw/renault/1309e6ff71.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34131/wide/renault-megane-iii-berline-occasion-2015-face-capot.jpg -
as ../../Dataset/Raw/renault/99c91094d8.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34113/wide/renault-clio-iv-occasion-2015-face-capot.jpg - as ../../D
ataset/Raw/renault/5ccf1625ca.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34106/wide/renault-captur-occasion-2015-face-capot.jpg - as ../../Da
taset/Raw/renault/0418f0bc8e.jpg
```

Entrée [8]:
```
download_brand_angle_img('VOLKSWAGEN', 'face-capot', '../../Dataset/Raw/volkswagen')
```

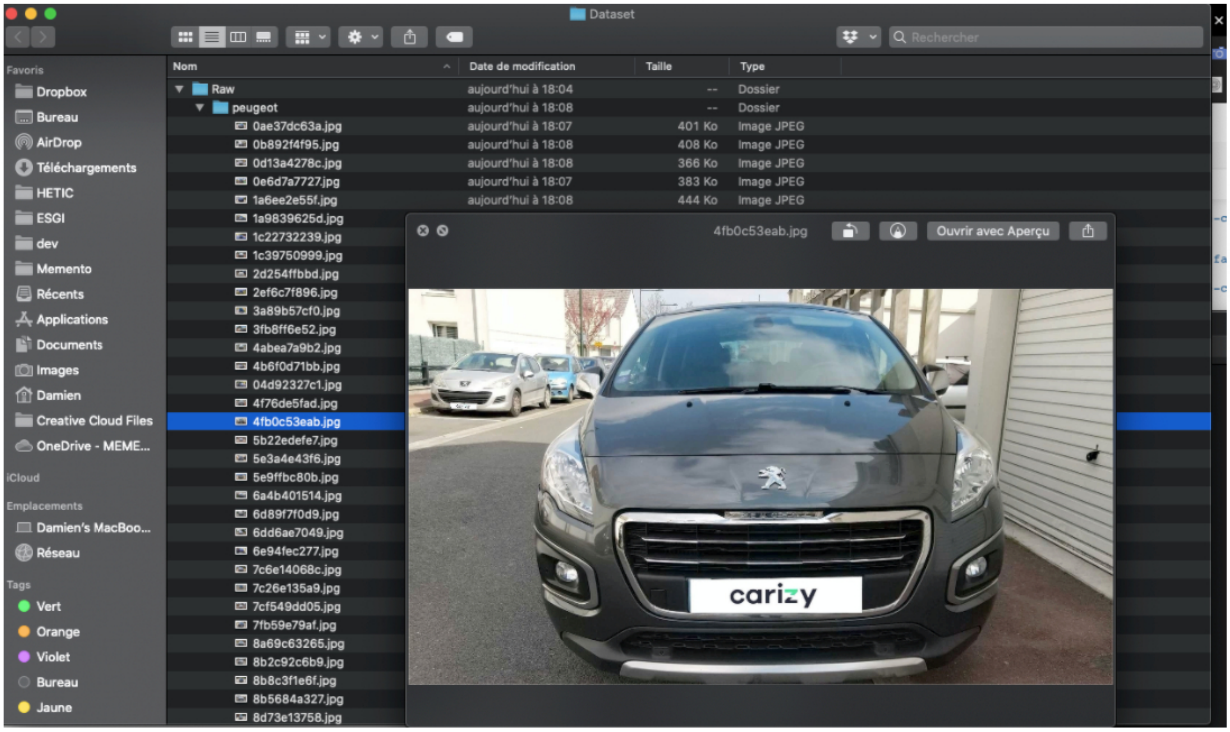
```
SUCCESS - saved https://cdn.carizy.com/carphotos/34169/wide/volkswagen-golf-occasion-2017-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/8018693f4f.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33959/wide/volkswagen-polo-occasion-2014-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/0b052bc4ad.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34029/wide/volkswagen-golf-occasion-2015-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/967cca70b5.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34031/wide/volkswagen-tiguan-occasion-2014-face-capot.jpg - as
../../Dataset/Raw/volkswagen/123284bb07.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33990/wide/volkswagen-touran-occasion-2012-face-capot.jpg - as
../../Dataset/Raw/volkswagen/f8b23173c9.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33973/wide/volkswagen-golf-occasion-2011-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/81378faff0.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/32855/wide/volkswagen-tiguan-occasion-2010-face-capot.jpg - as
../../Dataset/Raw/volkswagen/4059b8ad66.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33930/wide/volkswagen-polo-occasion-2012-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/582851287b.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33721/wide/volkswagen-polo-occasion-2010-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/1a86290004.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33847/wide/volkswagen-eos-occasion-2013-face-capot.jpg - as ../../Da
taset/Raw/volkswagen/869d74cdcd.jpg
```

```
SUCCESS - saved https://cdn.carizy.com/carphotos/33835/wide/volkswagen-golf-occasion-2012-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/394631a248.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33817/wide/volkswagen-golf-occasion-2012-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/4b7a6ae6b5.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33767/wide/volkswagen-polo-occasion-2013-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/5ae8b829bf.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33788/wide/volkswagen-polo-occasion-2014-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/905f6a91c1.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33784/wide/volkswagen-golf-sw-occasion-2016-face-capot.jpg - as
../../Dataset/Raw/volkswagen/998f867187.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33717/wide/volkswagen-polo-occasion-2014-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/3e2e32bd85.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33454/wide/volkswagen-golf-occasion-2016-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/92e64fd9fb.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/31193/wide/volkswagen-golf-occasion-2015-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/822f17843f.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/30242/wide/volkswagen-passat-occasion-2014-face-capot.jpg - as
../../Dataset/Raw/volkswagen/ff80abc3e5.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/30955/wide/volkswagen-passat-sw-occasion-2013-face-capot.jpg - as
../../Dataset/Raw/volkswagen/081e36df13.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/29934/wide/volkswagen-coccinelle-occasion-2017-face-capot.jpg - as
../../Dataset/Raw/volkswagen/641932f6bd.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/29965/wide/volkswagen-passat-occasion-2014-face-capot.jpg - as
../../Dataset/Raw/volkswagen/5b0265dfd2.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/29121/wide/volkswagen-golf-occasion-2017-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/61467f0128.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13517/wide/volkswagen-tiguan-occasion-2015-face-capot.jpg - as
../../Dataset/Raw/volkswagen/9257f695d6.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13324/wide/volkswagen-golf-occasion-2017-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/4fe1ec3daa.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13229/wide/volkswagen-golf-sw-occasion-2013-face-capot.jpg - as
../../Dataset/Raw/volkswagen/5a640d1a40.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13318/wide/volkswagen-golf-occasion-2014-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/37e4b65361.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13310/wide/volkswagen-golf-occasion-2015-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/d7c49e36dc.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13295/wide/volkswagen-tiguan-occasion-2014-face-capot.jpg - as
../../Dataset/Raw/volkswagen/0b9d6264a5.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13174/wide/volkswagen-tiguan-occasion-2015-face-capot.jpg - as
../../Dataset/Raw/volkswagen/0d74d3559f.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13073/wide/volkswagen-golf-occasion-2014-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/e9902ee6d6.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/12929/wide/volkswagen-golf-sportsvan-occasion-2015-face-capot.jpg -
as ../../Dataset/Raw/volkswagen/d0956ec363.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/12834/wide/volkswagen-polo-occasion-2013-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/1d77b1eed8.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/12816/wide/volkswagen-golf-sw-occasion-2014-face-capot.jpg - as
../../Dataset/Raw/volkswagen/eaa0e53046.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/12741/wide/volkswagen-tiguan-occasion-2012-face-capot.jpg - as
../../Dataset/Raw/volkswagen/14ca47cac8.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/12496/wide/volkswagen-sharan-occasion-2014-face-capot.jpg - as
../../Dataset/Raw/volkswagen/57b7d7c1a1.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/12425/wide/volkswagen-tiguan-occasion-2011-face-capot.jpg - as
../../Dataset/Raw/volkswagen/4d90ea878e.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/12411/wide/volkswagen-touran-occasion-2014-face-capot.jpg - as
../../Dataset/Raw/volkswagen/724743f374.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/12295/wide/volkswagen-tiguan-occasion-2013-face-capot.jpg - as
../../Dataset/Raw/volkswagen/2b281d0e1e.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/11090/wide/volkswagen-tiguan-occasion-2013-face-capot.jpg - as
../../Dataset/Raw/volkswagen/a91d110298.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/34015/wide/volkswagen-cc-occasion-2015-face-capot.jpg - as ../../Dat
aset/Raw/volkswagen/6a3ffbdba9.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33922/wide/volkswagen-golf-plus-occasion-2012-face-capot.jpg - as
../../Dataset/Raw/volkswagen/1f73243d7a.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33907/wide/volkswagen-polo-occasion-2014-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/10bdd87859.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33876/wide/volkswagen-polo-occasion-2015-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/bab0fa4f99.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/32856/wide/volkswagen-golf-occasion-2014-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/f8dfd2987c.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/32296/wide/volkswagen-touran-occasion-2012-face-capot.jpg - as
../../Dataset/Raw/volkswagen/c5bc537f03.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33920/wide/volkswagen-tiguan-occasion-2019-face-capot.jpg - as
../../Dataset/Raw/volkswagen/a4c5905499.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33908/wide/volkswagen-passat-occasion-2016-face-capot.jpg - as
../../Dataset/Raw/volkswagen/17029bba0d.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/33173/wide/volkswagen-polo-occasion-2012-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/71456b0f47.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/32864/wide/volkswagen-tiguan-occasion-2017-face-capot.jpg - as
../../Dataset/Raw/volkswagen/46d004d98c.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/31563/wide/volkswagen-polo-occasion-2014-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/3751166a1c.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/31185/wide/volkswagen-polo-occasion-2016-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/5b7ee2c239.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/30394/wide/volkswagen-up-occasion-2015-face-capot.jpg - as ../../Dat
aset/Raw/volkswagen/39fc034c6d.jpg

SUCCESS - saved https://cdn.carizy.com/carphotos/29948/wide/volkswagen-tiguan-occasion-2012-face-capot.jpg - as
../../Dataset/Raw/volkswagen/7d25477ac8.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/28386/wide/volkswagen-up-occasion-2018-face-capot.jpg - as ../../Dat
aset/Raw/volkswagen/b4449584ec.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/27632/wide/volkswagen-polo-occasion-2017-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/9fbcd79385.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13571/wide/volkswagen-polo-occasion-2018-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/14e0fc376a.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13532/wide/volkswagen-polo-occasion-2018-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/93a7d41ee4.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13446/wide/volkswagen-golf-occasion-2010-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/b48c39228c.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13520/wide/volkswagen-golf-occasion-2011-face-capot.jpg - as ../../D
```

```
ataset/Raw/volkswagen/7497131dca.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13450/wide/volkswagen-touran-occasion-2017-face-capot.jpg - as
../../Dataset/Raw/volkswagen/c52243f499.jpg
SUCCESS - saved https://cdn.carizy.com/carphotos/13471/wide/volkswagen-golf-occasion-2018-face-capot.jpg - as ../../D
ataset/Raw/volkswagen/b998e6e23d.jpg
```

Now we have a qualified dataset with lot of pictures ready to be trated.



2.3 Data exploration & analysis

In this part we will be exploring our dataset and analyze the repartition of each brand in order to be able to evaluate the accurcay of our future model.

Entrée [3]:
```
# Firstly import all important libraries even for further work
import tensorflow.keras.layers as Layers
import tensorflow.keras.activations as Actications
import tensorflow.keras.models as Models
import tensorflow.keras.optimizers as Optimizer
import tensorflow.keras.metrics as Metrics
import tensorflow.keras.utils as Utils
import tensorflow.keras.preprocessing
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.utils import model_to_dot
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
%matplotlib inline
import matplotlib.pyplot as plot
import cv2
import numpy as np
import seaborn as sns
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix as CM
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from random import randint
from IPython.display import SVG
import matplotlib.gridspec as gridspec
import pandas as pd
import imageio # pip install imageio

print('Dependencies imported')
```

```
Dependencies imported
```

Now, We need to prepare our data to feed into the model. Following is the gernalized function used to extract data from the directory.

Entrée [2]:
```
def get_images(directory):
    Images = []
    Labels = []  # 0 for Peugeot, 1 for Renault, 2 for Volkswagen
    label = 0

    for labels in os.listdir(directory): #Main Directory where each class label is present as folder name.
        if labels == 'peugeot': #Folder contain peugeot picture get label 0
            label = 0
```

```
        elif labels == 'renault':
            label = 1
        elif labels == 'volkswagen':
            label = 2


        for image_file in os.listdir(directory+labels): #Extracting the file name of the image from Class Label folder
            image = cv2.imread(directory+labels+r'/'+image_file) #Reading the image (OpenCV)
            image = cv2.resize(image,(128,128)) #Resize the image, Some images are different sizes. (Resizing is very
            Images.append(image)
            Labels.append(label)

    return shuffle(Images,Labels,random_state=817328462) #Shuffle the dataset

def get_classlabel(class_code):
    labels = {0:'peugeot', 1:'renault', 2:'volkswagen'}

    return labels[class_code]
```

Lets find shape of our traing data. The Training data is in shape of (Number of Training Images, Width of image, Height of image, Channel of image). This shape is very important. If we did not resize the images to same size. It should be (No. of images,) shape. So, using this shape we cant feed the images to the model

Entrée [3]:
```
Images, Labels = get_images('../../Dataset/Raw/') #Extract the training images from the folders.

Images = np.array(Images) #converting the list of images to numpy array.
Labels = np.array(Labels)
print('Dataset loaded')#If error remove all .ds stores in project => find . -name '.DS_Store' -type f -delete
```

```
Dataset loaded
```

Entrée [4]:
```
print("Shape of Images:",Images.shape)
print("Shape of Labels:",Labels.shape)
```

```
Shape of Images: (389, 128, 128, 3)
Shape of Labels: (389,)
```

So we have **389** pictures sized of 128 x 128 px with a 3 dimension for RGB. These are already **normalized** to a 128x128 matrix.
Let's have a look to some random images of our dataset just for pleasure:

For any image specific classification, clustering, etc. transforms we'll want to collapse spatial dimensions so that we have a matrix of pixels by color channels.

Entrée [5]:
```
first = plot.imread('../../Dataset/Raw/peugeot/0ae37dc63a.jpg')
dims = np.shape(first)
print(dims)
np.min(first), np.max(first)
# shape without redimensionning
```

```
(1080, 1920, 3)
```

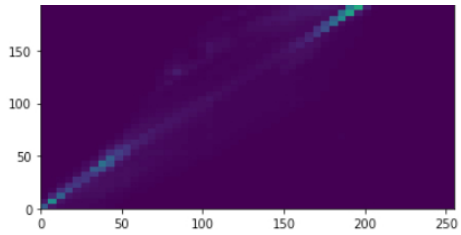
Out[5]:
```
(0, 255)
```

For any image specific classification, clustering, etc. transforms we'll want to collapse spatial dimensions so that we have a matrix of pixels by color channels.

Entrée [6]:
```
pixel_matrix = np.reshape(first, (dims[0] * dims[1], dims[2]))
print(np.shape(pixel_matrix))
plot.hist2d(pixel_matrix[:,1], pixel_matrix[:,2], bins=(50,50))
```

```
(2073600, 3)
```

Out[6]:
```
(array([[21240.,  4811.,   426., ...,     0.,     0.,     0.],
        [ 4265., 36456.,  7012., ...,     0.,     0.,     0.],
        [ 1470.,  3954., 23295., ...,     0.,     0.,     0.],
        ...,
        [    0.,     0.,     0., ...,  4939.,  4534.,  1500.],
        [    0.,     0.,     0., ...,  2841.,  8856.,  6802.],
        [    0.,     0.,     0., ...,   558.,  5571., 74550.]]),
 array([  0. ,   5.1,  10.2,  15.3,  20.4,  25.5,  30.6,  35.7,  40.8,
         45.9,  51. ,  56.1,  61.2,  66.3,  71.4,  76.5,  81.6,  86.7,
         91.8,  96.9, 102. , 107.1, 112.2, 117.3, 122.4, 127.5, 132.6,
        137.7, 142.8, 147.9, 153. , 158.1, 163.2, 168.3, 173.4, 178.5,
        183.6, 188.7, 193.8, 198.9, 204. , 209.1, 214.2, 219.3, 224.4,
        229.5, 234.6, 239.7, 244.8, 249.9, 255. ]),
 array([  0. ,   5.1,  10.2,  15.3,  20.4,  25.5,  30.6,  35.7,  40.8,
         45.9,  51. ,  56.1,  61.2,  66.3,  71.4,  76.5,  81.6,  86.7,
         91.8,  96.9, 102. , 107.1, 112.2, 117.3, 122.4, 127.5, 132.6,
        137.7, 142.8, 147.9, 153. , 158.1, 163.2, 168.3, 173.4, 178.5,
        183.6, 188.7, 193.8, 198.9, 204. , 209.1, 214.2, 219.3, 224.4,
        229.5, 234.6, 239.7, 244.8, 249.9, 255. ]),
 <matplotlib.collections.QuadMesh at 0x7fd1227cc550>)
```



Observe basic property of one image then we we will compute dataset global values.

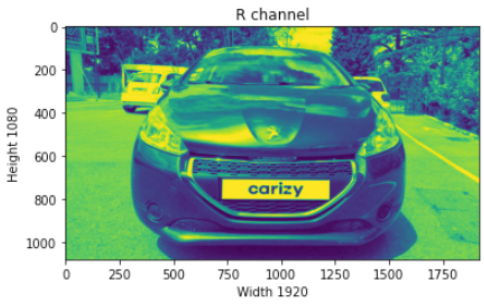
Entrée [7]:
```
plot.figure(figsize = (5,5))
plot.imshow(first)
print('Type of the image : ' , type(first))
print('Shape of the image : {}'.format(first.shape))
print('Image Hight {}'.format(first.shape[0]))
print('Image Width {}'.format(first.shape[1]))
print('Dimension of Image {}'.format(first.ndim))
print('Image size {}'.format(first.size))
print('Maximum RGB value in this image {}'.format(first.max()))
print('Minimum RGB value in this image {}'.format(first.min()))
```

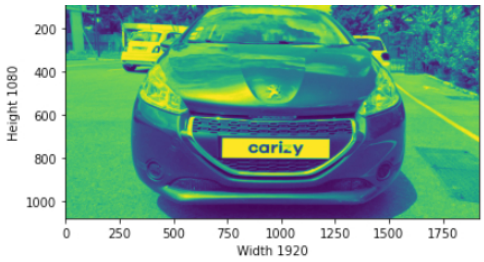
```
Type of the image :  <class 'numpy.ndarray'>
Shape of the image : (1080, 1920, 3)
Image Hight 1080
Image Width 1920
Dimension of Image 3
Image size 6220800
Maximum RGB value in this image 255
Minimum RGB value in this image 0
```

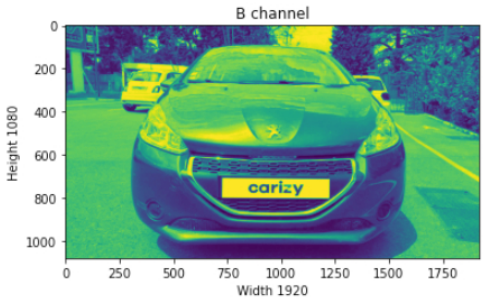
The shape of the ndarray shows that it is a three-layered matrix. The first two numbers here are length and width, and the third number (i.e. 3) is for three layers: Red, Green, Blue. So, if we calculate the size of an RGB image, the total size will be counted as height x width x 3

Okay, now let's take a quick view of each channel in the whole image.

Entrée [8]:
```
plot.title('R channel')
plot.ylabel('Height {}'.format(first.shape[0]))
plot.xlabel('Width {}'.format(first.shape[1]))
plot.imshow(first[ : , : , 0])
plot.show()
plot.title('G channel')
plot.ylabel('Height {}'.format(first.shape[0]))
plot.xlabel('Width {}'.format(first.shape[1]))
plot.imshow(first[ : , : , 1])
plot.show()
plot.title('B channel')
plot.ylabel('Height {}'.format(first.shape[0]))
plot.xlabel('Width {}'.format(first.shape[1]))
plot.imshow(first[ : , : , 2])
plot.show()
```





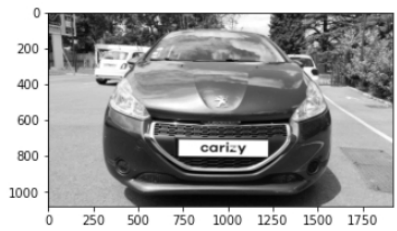


Splitting layers We are looking for an intersting output on a picture to see if it could be better use for model training

Entrée [9]:
```
fig, ax = plot.subplots(nrows = 1, ncols=3, figsize=(15,5))
for c, ax in zip(range(3), ax):
     # create zero matrix
     split_img = np.zeros(first.shape, dtype="uint8")
     # 'dtype' by default: 'numpy.float64'  # assing each channel
     split_img[ :, :, c] = first[ :, :, c] # display each channel
     ax.imshow(split_img)
```

Not very interesting. Maybe we could look for a grayscale image. It coulb be used later for computational matters.

Entrée [10]:
```
gray = lambda rgb : np.dot(rgb[... , :3] , [0.299 , 0.587, 0.114])
gray = gray(first)
plot.figure( figsize = (5,5))
plot.imshow(gray, cmap = plot.get_cmap(name = 'gray'))
plot.show()
```



The car brand is still recognizable and the logo also. Moreover it reduces the dimension of the np array and it could be a clear advantage to train the model.

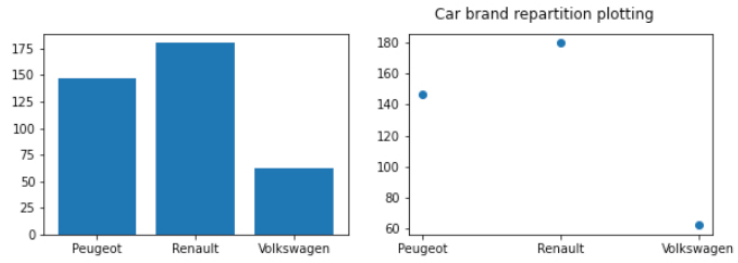
Let's visualize a few image of the dataset just for pleasure.

Entrée [11]:
```
f,ax = plot.subplots(5,5)
f.subplots_adjust(0,0,3,3)
for i in range(5):
    for j in range(5):
        rnd_number = randint(0,len(Images))
        ax[i,j].imshow(Images[rnd_number])
        ax[i,j].set_title(get_classlabel(Labels[rnd_number]))
        ax[i,j].axis('off')
```

Data exploration We are now going to visualize the repartition of each car brand with a bar and scatter chart.

Entrée [12]:
```
unique, counts = np.unique(Labels, return_counts=True)
occurences = np.asarray((unique, counts)).T
names = ['Peugeot', 'Renault', 'Volkswagen']
values = [occurences[0][1], occurences[1][1], occurences[2][1]]
total = Labels.size
plot.figure(figsize=(15, 3))

plot.subplot(131)
plot.bar(names, values)
plot.subplot(132)
plot.scatter(names, values)
plot.suptitle('Car brand repartition plotting')
plot.show()
print("Peugeot : " + str(occurences[0][1]) + " | " + str(int(occurences[0][1] / total * 10000) / 100) + "%")
print("Renault : " + str(occurences[1][1]) + " | " + str(int(occurences[1][1] / total * 10000) / 100) + "%")
print("Volkswagen : " + str(occurences[2][1]) + " | " + str(int(occurences[2][1] / total * 10000) / 100) + "%")
```




```
Peugeot : 147 | 37.78%
Renault : 180 | 46.27%
Volkswagen : 62 | 15.93%
```

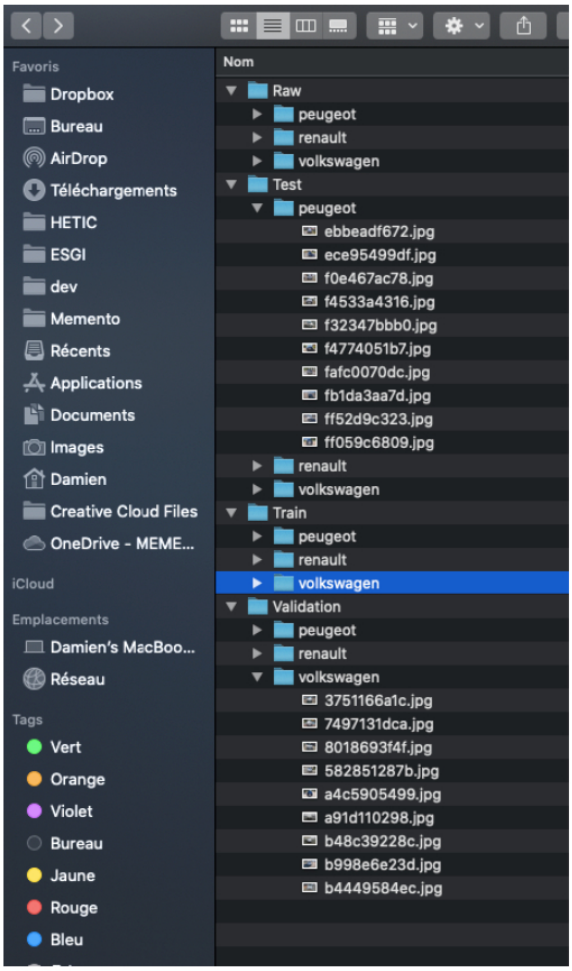
Regarding those results we are now able to estimate our model efficiency.
If the model predict peugeot less than the random (46.5%) then it won't be good enough. Since it could be prediciting renault each time.

3. Data pre-processing

Since we have already loaded the data in the Images class and the dataset was already normliazed to 128x128 matrix we will the validation and train set and then label encode the data.

3.1 Split training & validation set

I choosed to split the train set in two parts : a small fraction (20%) became the validation set which the model is evaluated and the rest (80%) is used to train the model. This the safest and classical splitting values.
Moreover i will take **10 pictures** in each brand dataset to put it in a test set directly in the folders.



Entrée [13]:
```
# parameters variable

train_dir = "../../Dataset/Train"
validation_dir = "../../Dataset/Validation"
target_width = 128
target_height = 128
color_mode = 'rgb'
target_size = (target_width, target_height)
image_generator = ImageDataGenerator(rescale=1./255)

# Set train data
train_data = image_generator.flow_from_directory(
        train_dir,
        target_size=target_size,
        color_mode=color_mode,
    )
#set Validation data
validation_data = image_generator.flow_from_directory(
    validation_dir,
    target_size=target_size,
    color_mode=color_mode
)

train_data.batch_size = train_data.n
(x, y) = train_data.next()
train_true_label = np.argmax(y, axis=1)
```

```
Found 289 images belonging to 3 classes.
Found 70 images belonging to 3 classes.
```

Perfec the dataset is loaded and and resized to 128x128 pictures

4. Neural networks training

We are now going to start training our first neural network by using three different models:

- Linear Model
- Multi Layer Perceptron

- Multi Layer Perceptron
- Support Vector machine
- Convolutional Neural Network

Entrée [4]:
```
# Necessary classes for building a correct model and neural network
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPool2D, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.preprocessing import StandardScaler
from tensorflow.keras import regularizers
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np
```

4.1 Linear Model

We are now going to build the linear classfication model which will be represented by neural network without any hidden layer.

Entrée [15]:
```
train_dir = "../../Dataset/Train/"
validation_dir = "../../Dataset/Validation/"
target_width = 128
target_height = 128
color_mode = 'rgb'
target_size = (target_width, target_height)
image_generator = ImageDataGenerator(rescale=1./255)

# Set train data
train_data = image_generator.flow_from_directory(
        train_dir,
        target_size=target_size,
        color_mode=color_mode,
    )
#set Validation data
validation_data = image_generator.flow_from_directory(
    validation_dir,
    target_size=target_size,
    color_mode=color_mode)
```

```
Found 289 images belonging to 3 classes.
Found 70 images belonging to 3 classes.
```

Entrée [25]:
```
linear_model = Sequential()
linear_model.add(Flatten(input_shape=(128,128,3)))
linear_model.add(Dense(3))
linear_model.add(Activation('softmax'))

linear_model.compile(optimizer='SGD',
              loss=tensorflow.keras.losses.mean_squared_error,
              metrics=['accuracy'])
linear_model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 49152)             0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 147459
_________________________________________________________________
activation_3 (Activation)    (None, 3)                 0
=================================================================
Total params: 147,459
Trainable params: 147,459
Non-trainable params: 0
_________________________________________________________________
```

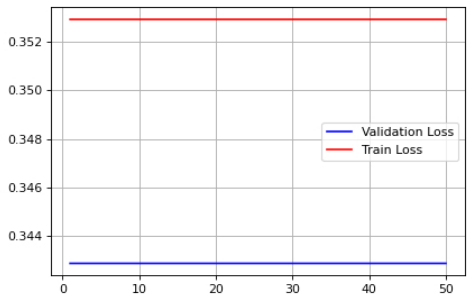
Entrée [33]:
```
# Train the model
history = linear_model.fit(train_data, epochs=50, validation_data=validation_data,
              callbacks=[tensorflow.keras.callbacks.TensorBoard(
                  log_dir="../Results/Logs/basic_linear_sgd_mse_rgb")])
```

```
Epoch 1/50
10/10 [==============================] - 10s 1s/step - loss: 0.3529 - accuracy: 0.4706 - val_loss: 0.3429 - val_accur
acy: 0.4857
Epoch 2/50
10/10 [==============================] - 10s 1s/step - loss: 0.3529 - accuracy: 0.4706 - val_loss: 0.3429 - val_accur
acy: 0.4857
Epoch 3/50
10/10 [==============================] - 10s 1s/step - loss: 0.3529 - accuracy: 0.4706 - val_loss: 0.3429 - val_accur
acy: 0.4857
Epoch 4/50
10/10 [==============================] - 10s 989ms/step - loss: 0.3529 - accuracy: 0.4706 - val_loss: 0.3429 - val_ac
curacy: 0.4857
Epoch 5/50
10/10 [==============================] - 10s 1s/step - loss: 0.3529 - accuracy: 0.4706 - val_loss: 0.3429 - val_accur
acy: 0.4857
Epoch 6/50
10/10 [==============================] - 10s 989ms/step - loss: 0.3529 - accuracy: 0.4706 - val_loss: 0.3429 - val_ac
curacy: 0.4857
Epoch 7/50
10/10 [==============================] - 10s 990ms/step - loss: 0.3529 - accuracy: 0.4706 - val_loss: 0.3429 - val_ac
```

Entrée [28]:
```
#utility function to show loss in interactive plot
%matplotlib notebook
import matplotlib.pyplot as plot
def plot_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plot.legend()
    plot.grid()
    fig.canvas.draw()
```

Entrée [36]:
```
fig, ax = plot.subplots(1,1)
ax.set_xlabel = "epoch"
ax.set_ylabel = "Mean Squared Error Loss"
x= list(range(1, 50+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_dynamic(x, vy, ty, ax)
```

```
<IPython.core.display.Javascript object>
```



Entrée [38]:
```
 train_data.batch_size = train_data.n
(x, y) = train_data.next()
preds = linear_model.predict(x)
train_preds_label = np.argmax(preds, axis=1)
train_true_label = np.argmax(y, axis=1)

validation_data.batch_size = validation_data.n
(x, y) = validation_data.next()
preds = linear_model.predict(x)
validation_preds_label = np.argmax(preds, axis=1)
validation_true_label = np.argmax(y, axis=1)

print("Training Set Confusion Matrix : ")
print(metrics.confusion_matrix(train_true_label, train_preds_label))

print("Validation Set Confusion Matrix : ")
print(metrics.confusion_matrix(validation_true_label, validation_preds_label))
```

```
Training Set Confusion Matrix :
[[  0 110   0]
 [  0 136   0]
 [  0  43   0]]
Validation Set Confusion Matrix :
[[ 0 27  0]
 [ 0 34  0]
 [ 0  9  0]]
```

The model is converging too fast to full Renault recognition. We need a more complex neural network.

4.2 Multi Layer Pereceptron

Let's try now with a second type of neural network. We hope for better results. We will be able tu use the same validation and training set in order to compare the results.

Entrée [42]:
```
# Build basic MLP model
mlp_model = Sequential()
mlp_model.add(Flatten(input_shape=(128,128,3)))
mlp_model.add(Dense(128, activation='tanh'))
mlp_model.add(Dense(128, activation='tanh'))
mlp_model.add(Dense(3, activation='softmax'))
mlp_model.compile(loss=tensorflow.keras.losses.mean_squared_error, metrics=['accuracy'])
mlp_model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_6 (Flatten)          (None, 49152)             0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               6291584
_________________________________________________________________
dense_8 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 387
=================================================================
```

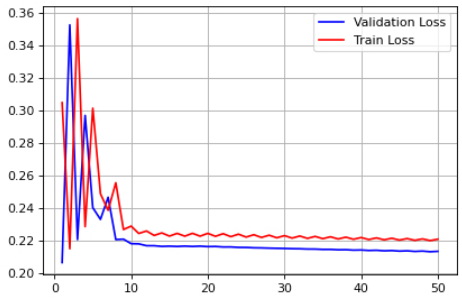
```
Total params: 6,308,483
Trainable params: 6,308,483
Non-trainable params: 0
_________________________________________________________________
```

Entrée [43]:
```
# Train the model
history = mlp_model.fit(train_data, epochs=50, validation_data=validation_data,
                  callbacks=[tensorflow.keras.callbacks.TensorBoard(
                      log_dir="../Results/Logs/basic_mlp_128x128_rgb_MSErr")])
```

```
1/1 [==============================] - 4s 4s/step - loss: 0.2151 - accuracy: 0.4706 - val_loss: 0.3525 - val_accurac
y: 0.3857
Epoch 3/50
1/1 [==============================] - 4s 4s/step - loss: 0.3564 - accuracy: 0.3806 - val_loss: 0.2208 - val_accurac
y: 0.4857
Epoch 4/50
1/1 [==============================] - 4s 4s/step - loss: 0.2287 - accuracy: 0.4706 - val_loss: 0.2970 - val_accurac
y: 0.3857
Epoch 5/50
1/1 [==============================] - 4s 4s/step - loss: 0.3014 - accuracy: 0.3806 - val_loss: 0.2404 - val_accurac
y: 0.4857
Epoch 6/50
1/1 [==============================] - 5s 5s/step - loss: 0.2489 - accuracy: 0.4706 - val_loss: 0.2332 - val_accurac
y: 0.3857
Epoch 7/50
1/1 [==============================] - 4s 4s/step - loss: 0.2387 - accuracy: 0.3806 - val_loss: 0.2468 - val_accurac
y: 0.4857
Epoch 8/50
1/1 [==============================] - 4s 4s/step - loss: 0.2556 - accuracy: 0.4706 - val_loss: 0.2208 - val_accurac
y: 0.3857
Epoch 9/50
```

Entrée [44]:
```
fig, ax = plot.subplots(1,1)
ax.set_xlabel = "epoch"
ax.set_ylabel = "Mean Squared Error Loss"
x= list(range(1, 50+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_dynamic(x, vy, ty, ax)
```

```
<IPython.core.display.Javascript object>
```



Entrée [45]:
```
train_data.batch_size = train_data.n
(x, y) = train_data.next()
preds = linear_model.predict(x)
train_preds_label = np.argmax(preds, axis=1)
train_true_label = np.argmax(y, axis=1)

validation_data.batch_size = validation_data.n
(x, y) = validation_data.next()
preds = linear_model.predict(x)
validation_preds_label = np.argmax(preds, axis=1)
validation_true_label = np.argmax(y, axis=1)

print("Training Set Confusion Matrix : ")
print(metrics.confusion_matrix(train_true_label, train_preds_label))

print("Validation Set Confusion Matrix : ")
print(metrics.confusion_matrix(validation_true_label, validation_preds_label))
```

```
Training Set Confusion Matrix :
[[  0 110   0]
 [  0 136   0]
 [  0  43   0]]
Validation Set Confusion Matrix :
[[ 0 27  0]
 [ 0 34  0]
 [ 0  9  0]]
```

The model is underfitting and does not learn we want to try with a greyscale dataset since color can be a bad thing for features extraction for the model.

The computing is too long so we want to try reduce the input size. Treating data in greyscale seems a good idea. The feature of a car brand is not in it's color but more in it it's shape. In addition since we don't have every car color the coloration can give the models bad intuitions on color.

Entrée [46]:
```
#Let's generate the grayscale dataset
```

```
train_dir = "../../Dataset/Train"
validation_dir = "../../Dataset/Validation"
target_width = 128
target_height = 128
color_mode = 'grayscale'
target_size = (target_width, target_height)
image_generator = ImageDataGenerator(rescale=1./255)

# Set train data
train_data = image_generator.flow_from_directory(
        train_dir,
        target_size=target_size,
        color_mode=color_mode,
        #classes=['peugeot', 'renault', 'volkswagen']
    )
#set Validation data
validation_data = image_generator.flow_from_directory(
    validation_dir,
    target_size=target_size,
    color_mode=color_mode
    #classes=['peugeot', 'renault', 'volkswagen']
)

train_data.batch_size = train_data.n
(x, y) = train_data.next()
train_true_label = np.argmax(y, axis=1)
print(train_data.next()[0].shape)
```

```
Found 289 images belonging to 3 classes.
Found 70 images belonging to 3 classes.
(289, 128, 128, 1)
```

Entrée [47]:

```
mlp_model = Sequential()
mlp_model.add(Flatten(input_shape=(128,128, 1)))
mlp_model.add(Dense(512, activation='relu'))
mlp_model.add(Dense(512, activation='relu'))
mlp_model.add(Dense(512, activation='relu'))
mlp_model.add(Dense(3, activation='softmax'))

# save the best model according to validation data set accuracy
checkpoint = ModelCheckpoint('../../App/backend/models/mlp_m.h5', monitor='val_accuracy', verbose=1, save_best_only=T

# compile the model
mlp_model.compile(optimizer='SGD', loss=tensorflow.keras.losses.categorical_crossentropy, metrics=['accuracy'])
mlp_model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_7 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_10 (Dense)             (None, 512)               8389120
_________________________________________________________________
dense_11 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_12 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_13 (Dense)             (None, 3)                 387
=================================================================
Total params: 8,553,731
Trainable params: 8,553,731
Non-trainable params: 0
_________________________________________________________________
```

Entrée [48]:

```
history = mlp_model.fit(train_data, batch_size = 32, epochs=50, validation_data=validation_data,
              callbacks=[tensorflow.keras.callbacks.TensorBoard(
                  log_dir="../Results/Logs/triple512_sgd_relu_simple_mlp_greyscale_categorical-crossentropy"), checkp
```

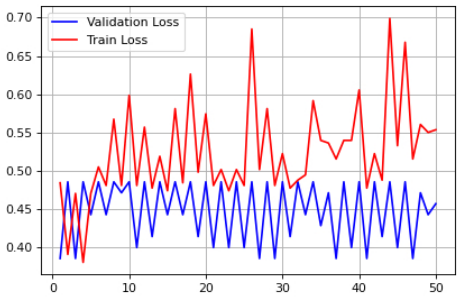
```
1/1 [==============================] - ETA: 0s - loss: 0.8297 - accuracy: 0.5917
Epoch 00034: val_accuracy did not improve from 0.48571
1/1 [==============================] - 3s 3s/step - loss: 0.8297 - accuracy: 0.5917 - val_loss: 1.0160 - val_accurac
y: 0.4857
Epoch 35/50
1/1 [==============================] - ETA: 0s - loss: 0.8251 - accuracy: 0.5398
Epoch 00035: val_accuracy did not improve from 0.48571
1/1 [==============================] - 3s 3s/step - loss: 0.8251 - accuracy: 0.5398 - val_loss: 1.0238 - val_accurac
y: 0.4286
Epoch 36/50
1/1 [==============================] - ETA: 0s - loss: 0.8394 - accuracy: 0.5363
Epoch 00036: val_accuracy did not improve from 0.48571
1/1 [==============================] - 3s 3s/step - loss: 0.8394 - accuracy: 0.5363 - val_loss: 1.0595 - val_accurac
y: 0.4714
Epoch 37/50
1/1 [==============================] - ETA: 0s - loss: 0.8637 - accuracy: 0.5156
Epoch 00037: val_accuracy did not improve from 0.48571
1/1 [==============================] - 3s 3s/step - loss: 0.8637 - accuracy: 0.5156 - val_loss: 1.0258 - val_accurac
y: 0.3857
Epoch 38/50
1/1 [==============================] - ETA: 0s - loss: 0.8286 - accuracy: 0.5398
```

Entrée [49]:

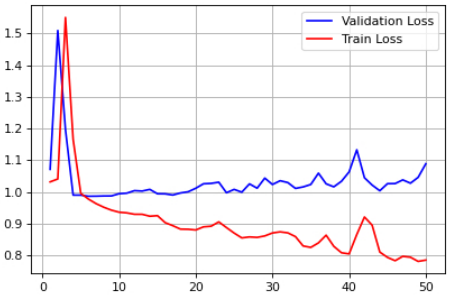
```
fig, ax = plot.subplots(1,1)
ax.set_xlabel = "epoch"
ax.set_ylabel = "Mean Squared Error Loss"
x= list(range(1, 50+1))
vy = history.history['val_accuracy']
ty = history.history['accuracy']
plot_dynamic(x, vy, ty, ax)
```

```
#Loss
fig, ax = plot.subplots(1,1)
ax.set_xlabel = "epoch"
ax.set_ylabel = "Mean Squared Error Loss"
x= list(range(1, 50+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_dynamic(x, vy, ty, ax)
```

```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```



Entrée [51]:
```
train_data.batch_size = train_data.n
(x, y) = train_data.next()
preds = mlp_model.predict(x)
train_preds_label = np.argmax(preds, axis=1)
train_true_label = np.argmax(y, axis=1)

validation_data.batch_size = validation_data.n
(x, y) = validation_data.next()
preds = mlp_model.predict(x)
validation_preds_label = np.argmax(preds, axis=1)
validation_true_label = np.argmax(y, axis=1)

print("Training Set Confusion Matrix : ")
print(CM(train_true_label, train_preds_label))

print("Validation Set Confusion Matrix : ")
print(CM(validation_true_label, validation_preds_label))
```

```
Training Set Confusion Matrix :
[[  1 108   1]
 [  0 136   0]
 [  0  25  18]]
Validation Set Confusion Matrix :
[[ 0 26  1]
 [ 0 32  2]
 [ 0  9  0]]
```

Entrée [ ]:
```
<div class="alert alert-block alert-info">
Since the dataset is not large enough (389 images) for 3 classes with lot of paramaters we awill try produce data augm
</div>
```

Data augmentation

In order to improve the datset that is not big enough for geenralization we will proceed to data augmentation:

- First we will change the zoom on each image since usually the center of the image is where the car is situated
- Then we will proceed to horizontal flip since it is not changing the car features
- A few degrees rotation won't change the brand features neither
- Will shift the image horizontally and vertically in order to create better strengh to the model

Entrée [33]:
```
#Let's generate the augmented dataset
train_dir = "../../Dataset/Train"
validation_dir = "../../Dataset/Validation"
target_width = 128
target_height = 128
color_mode = 'grayscale'
```

```
target_size = (target_width, target_height)
image_generator = ImageDataGenerator(rescale=1./255,
                                     zoom_range=0.30,#Zooming
                                     horizontal_flip=True,
                                     rotation_range=10,
                                     width_shift_range=0.2, #Horizontal shifting
                                     height_shift_range=0.2)#vertical shifting

# Set train data
train_data = image_generator.flow_from_directory(
        train_dir,
        target_size=target_size
```

```
        target_size=target_size,
        color_mode=color_mode,
        #classes=['peugeot', 'renault', 'volkswagen']
    )
#set Validation data
validation_data = image_generator.flow_from_directory(
    validation_dir,
    target_size=target_size,
    color_mode=color_mode
    #classes=['peugeot', 'renault', 'volkswagen']
)

train_data.batch_size = train_data.n
(x, y) = train_data.next()
train_true_label = np.argmax(y, axis=1)
```

```
Found 289 images belonging to 3 classes.
Found 70 images belonging to 3 classes.
```

Entrée [34]:
```
# Now that we have augmented the datset let's create the model
mlp_model = Sequential()
mlp_model.add(Flatten(input_shape=(128,128, 1)))
mlp_model.add(Dense(512, activation='relu'))
mlp_model.add(Dense(512, activation='relu'))
mlp_model.add(Dense(512, activation='relu'))
mlp_model.add(Dense(3, activation='softmax'))

# save the best model according to validation data set accuracy
checkpoint = ModelCheckpoint('../../App/backend/models/mlp_mod.h5', monitor='val_accuracy', verbose=1, save_best_only

# compile the model
mlp_model.compile(optimizer='SGD', loss=tensorflow.keras.losses.categorical_crossentropy, metrics=['accuracy'])
mlp_model.summary()
```

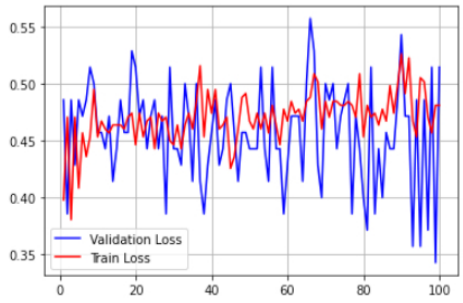
```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_6 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_7 (Dense)              (None, 512)               8389120
_________________________________________________________________
dense_8 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_9 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_10 (Dense)             (None, 3)                 1539
=================================================================
Total params: 8,915,971
Trainable params: 8,915,971
Non-trainable params: 0
_________________________________________________________________
```

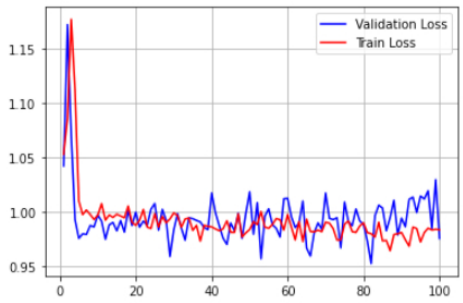
Entrée [35]:
```
history = mlp_model.fit(train_data, batch_size = 32, epochs=100, validation_data=validation_data,
              callbacks=[tensorflow.keras.callbacks.TensorBoard(
                  log_dir="../Results/Logs/mlp_fin"), checkpoint])
```

```
1/1 [==============================] - 3s 3s/step - loss: 0.9766 - accuracy: 0.4740 - val_loss: 0.9963 - val_accurac
y: 0.3857
Epoch 84/100
1/1 [==============================] - ETA: 0s - loss: 0.9903 - accuracy: 0.4637
Epoch 00084: val_accuracy did not improve from 0.55714
1/1 [==============================] - 3s 3s/step - loss: 0.9903 - accuracy: 0.4637 - val_loss: 1.0060 - val_accurac
y: 0.4429
Epoch 85/100
1/1 [==============================] - ETA: 0s - loss: 0.9731 - accuracy: 0.4775
Epoch 00085: val_accuracy did not improve from 0.55714
1/1 [==============================] - 3s 3s/step - loss: 0.9731 - accuracy: 0.4775 - val_loss: 1.0034 - val_accurac
y: 0.4000
Epoch 86/100
1/1 [==============================] - ETA: 0s - loss: 0.9735 - accuracy: 0.4671
Epoch 00086: val_accuracy did not improve from 0.55714
1/1 [==============================] - 3s 3s/step - loss: 0.9735 - accuracy: 0.4671 - val_loss: 0.9824 - val_accurac
y: 0.4571
Epoch 87/100
1/1 [==============================] - ETA: 0s - loss: 0.9640 - accuracy: 0.4983
Epoch 00087: val_accuracy did not improve from 0.55714
```

Entrée [41]:
```
fig, ax = plot.subplots(1,1)
ax.set_xlabel = "epoch"
ax.set_ylabel = "Accurcay"
x= list(range(1, 100+1))
vy = history.history['val_accuracy']
ty = history.history['accuracy']
plot_dynamic(x, vy, ty, ax)

#Loss
fig, ax = plot.subplots(1,1)
ax.set_xlabel = "epoch"
ax.set_ylabel = "Categorical crossentropy Loss"
x= list(range(1, 100+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_dynamic(x, vy, ty, ax)
```





Entrée [38]:
```
train_data.batch_size = train_data.n
(x, y) = train_data.next()
preds = mlp_model.predict(x)
train_preds_label = np.argmax(preds, axis=1)
train_true_label = np.argmax(y, axis=1)

validation_data.batch_size = validation_data.n
(x, y) = validation_data.next()
preds = mlp_model.predict(x)
validation_preds_label = np.argmax(preds, axis=1)
validation_true_label = np.argmax(y, axis=1)

print("Training Set Confusion Matrix : ")
print(CM(train_true_label, train_preds_label))

print("Validation Set Confusion Matrix : ")
print(CM(validation_true_label, validation_preds_label))
```

```
Training Set Confusion Matrix :
[[  8 102   0]
 [  9 127   0]
 [  1  42   0]]
Validation Set Confusion Matrix :
[[ 1 26  0]
 [ 0 34  0]
 [ 1  8  0]]
```

We have reached a 55% accuracy on the validation which is slightly better than random. The model predict too much Renault according the confusion matrices. That is due to its over representation in the dataset.

4.3 Support Vector Machine

We don't have good enough result will try adding a SVM (regulizer in Keras) to improve the model and reduce overfitting

Entrée [12]:
```
w that we have augmented the datset let's create the model
model = Sequential()

model.add(Flatten(input_shape=(128,128, 1)))
model.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.01)))# Kernel Regulizer is common Suppor
model.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(3, activation='softmax'))

ve the best model according to validation data set accuracy
kpoint = ModelCheckpoint('../../App/backend/models/svm_model.h5', monitor='val_accuracy', verbose=1, save_best_only=Tr

mpile the model
model.compile(optimizer='SGD', loss=tensorflow.keras.losses.mean_squared_error, metrics=['categorical_accuracy', 'accu
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               8389120
_________________________________________________________________
dense_5 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_6 (Dense)              (None, 512)               262656
```

```
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 1539
=================================================================
Total params: 8,915,971
Trainable params: 8,915,971
Non-trainable params: 0
_________________________________________________________________
```

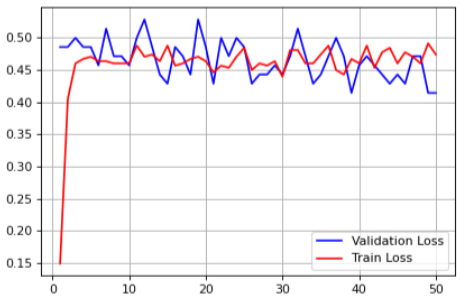
Entrée [13]:
```
history = svm_model.fit(train_data, batch_size = 32, epochs=50, validation_data=validation_data,
                callbacks=[tensorflow.keras.callbacks.TensorBoard(
                    log_dir="../Results/Logs/svm_triple512"), checkpoint])
```

```
Epoch 1/50
1/1 [==============================] - ETA: 0s - loss: 20.4940 - categorical_accuracy: 0.1488 - accuracy: 0.1488
Epoch 00001: val_accuracy improved from -inf to 0.48571, saving model to ../../App/backend/models/svm_model.h5
1/1 [==============================] - 3s 3s/step - loss: 20.4940 - categorical_accuracy: 0.1488 - accuracy: 0.1488 -
val_loss: 20.3941 - val_categorical_accuracy: 0.4857 - val_accuracy: 0.4857
Epoch 2/50
1/1 [==============================] - ETA: 0s - loss: 20.3977 - categorical_accuracy: 0.4048 - accuracy: 0.4048
Epoch 00002: val_accuracy did not improve from 0.48571
1/1 [==============================] - 3s 3s/step - loss: 20.3977 - categorical_accuracy: 0.4048 - accuracy: 0.4048 -
val_loss: 20.3780 - val_categorical_accuracy: 0.4857 - val_accuracy: 0.4857
Epoch 3/50
1/1 [==============================] - ETA: 0s - loss: 20.3753 - categorical_accuracy: 0.4602 - accuracy: 0.4602
Epoch 00003: val_accuracy improved from 0.48571 to 0.50000, saving model to ../../App/backend/models/svm_model.h5
1/1 [==============================] - 3s 3s/step - loss: 20.3753 - categorical_accuracy: 0.4602 - accuracy: 0.4602 -
val_loss: 20.3588 - val_categorical_accuracy: 0.5000 - val_accuracy: 0.5000
Epoch 4/50
1/1 [==============================] - ETA: 0s - loss: 20.3617 - categorical_accuracy: 0.4671 - accuracy: 0.4671
Epoch 00004: val_accuracy did not improve from 0.50000
1/1 [==============================] - 3s 3s/step - loss: 20.3617 - categorical_accuracy: 0.4671 - accuracy: 0.4671 -
val_loss: 20.3509 - val_categorical_accuracy: 0.4857 - val_accuracy: 0.4857
```

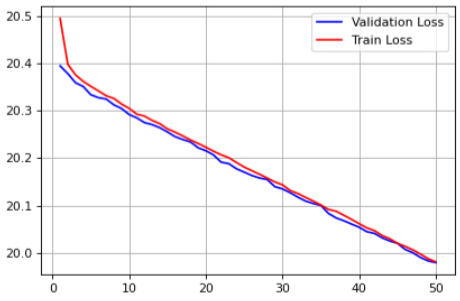
Entrée [14]:
```
fig, ax = plot.subplots(1,1)
ax.set_xlabel = "epoch"
ax.set_ylabel = "Accuracy"
x= list(range(1, 50+1))
vy = history.history['val_accuracy']
ty = history.history['accuracy']
plot_dynamic(x, vy, ty, ax)

#Loss
fig, ax = plot.subplots(1,1)
ax.set_xlabel = "epoch"
ax.set_ylabel = "Mean Squarred Error"
x= list(range(1, 50+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_dynamic(x, vy, ty, ax)
```

```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```



Entrée [16]:
```
train_data.batch_size = train_data.n
(x, y) = train_data.next()
preds = svm_model.predict(x)
train_preds_label = np.argmax(preds, axis=1)
train_true_label = np.argmax(y, axis=1)

validation_data.batch_size = validation_data.n
```

```
(x, y) = validation_data.next()
preds = svm_model.predict(x)
validation_preds_label = np.argmax(preds, axis=1)
validation_true_label = np.argmax(y, axis=1)

print("Training Set Confusion Matrix : ")
print(CM(train_true_label, train_preds_label))

print("Validation Set Confusion Matrix : ")
print(CM(validation_true_label, validation_preds_label))
```

```
Training Set Confusion Matrix :
[[ 25  85   0]
 [ 15 121   0]
 [  5  38   0]]
Validation Set Confusion Matrix :
[[ 5 22  0]
 [ 5 29  0]
 [ 2  7  0]]
```

The model has reach a 53% accuracy.

4.4 Convolutional Neural Network

The most efficient model for computer vision is know as convolutional neural network which will try to train to get some results.

Entrée [25]:
```
from tensorflow.keras.optimizers import SGD
opt = SGD(lr=0.01)

cnn_model = Sequential()
cnn_model.add(Conv2D(filters = 8, kernel_size = (3,3),padding = 'Same', activation ='relu',input_shape=((128,128,1)))
cnn_model.add(MaxPool2D())

cnn_model.add(Conv2D(filters = 16, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
cnn_model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
cnn_model.add(MaxPool2D())

# the model so far outputs 3D feature maps (height, width, features)
#On top of it we stick two fully-connected layers. We end the model with a single

cnn_model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
cnn_model.add(Dense(3))
cnn_model.add(Activation('softmax'))

# save the best model according to validation data set accuracy
checkpoint = ModelCheckpoint('../../App/backend/models/cnn_model.h5', monitor='val_accuracy', verbose=1, save_best_on

# compile the model
cnn_model.compile(optimizer=opt, loss=tensorflow.keras.losses.categorical_crossentropy, metrics=['accuracy'])
cnn_model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 128, 128, 8)       80
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 64, 64, 8)         0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 64, 64, 16)        1168
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 64, 64, 32)        4640
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 32, 32, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 32768)             0
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 98307
_________________________________________________________________
activation_5 (Activation)    (None, 3)                 0
=================================================================
Total params: 104,195
Trainable params: 104,195
Non-trainable params: 0
_________________________________________________________________
```

Entrée [42]:
```
#Proper epoch would be 50
history = cnn_model.fit_generator(train_data, epochs=20, validation_data=validation_data,
              callbacks=[tensorflow.keras.callbacks.TensorBoard(
                  log_dir="../Results/Logs/cnn_fin"), checkpoint])
```

```
Epoch 1/20
1/1 [==============================] - ETA: 0s - loss: 1.0056 - accuracy: 0.4671
Epoch 00001: val_accuracy did not improve from 0.55714
1/1 [==============================] - 4s 4s/step - loss: 1.0056 - accuracy: 0.4671 - val_loss: 0.9802 - val_accurac
y: 0.4857
```

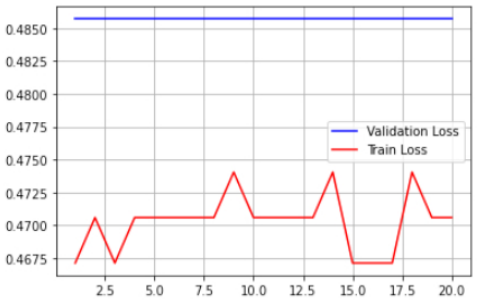
```
Epoch 2/20
1/1 [==============================] - ETA: 0s - loss: 1.0052 - accuracy: 0.4706
Epoch 00002: val_accuracy did not improve from 0.55714
1/1 [==============================] - 4s 4s/step - loss: 1.0052 - accuracy: 0.4706 - val_loss: 0.9771 - val_accurac
y: 0.4857
Epoch 3/20
1/1 [==============================] - ETA: 0s - loss: 1.0010 - accuracy: 0.4671
Epoch 00003: val_accuracy did not improve from 0.55714
1/1 [==============================] - 4s 4s/step - loss: 1.0010 - accuracy: 0.4671 - val_loss: 0.9838 - val_accurac
y: 0.4857
Epoch 4/20
1/1 [==============================] - ETA: 0s - loss: 1.0012 - accuracy: 0.4706
Epoch 00004: val_accuracy did not improve from 0.55714
1/1 [==============================] - 4s 4s/step - loss: 1.0012 - accuracy: 0.4706 - val_loss: 0.9883 - val_accurac
y: 0.4857
Epoch 5/20
1/1 [==============================] - ETA: 0s - loss: 1.0053 - accuracy: 0.4706
Epoch 00005: val_accuracy did not improve from 0.55714
1/1 [==============================] - 4s 4s/step - loss: 1.0053 - accuracy: 0.4706 - val_loss: 0.9802 - val_accurac
y: 0.4857
Epoch 6/20
1/1 [==============================] - ETA: 0s - loss: 1.0042 - accuracy: 0.4706
Epoch 00006: val_accuracy did not improve from 0.55714
1/1 [==============================] - 5s 5s/step - loss: 1.0042 - accuracy: 0.4706 - val_loss: 0.9922 - val_accurac
y: 0.4857
Epoch 7/20
1/1 [==============================] - ETA: 0s - loss: 1.0051 - accuracy: 0.4706
Epoch 00007: val_accuracy did not improve from 0.55714
1/1 [==============================] - 5s 5s/step - loss: 1.0051 - accuracy: 0.4706 - val_loss: 0.9801 - val_accurac
y: 0.4857
Epoch 8/20
1/1 [==============================] - ETA: 0s - loss: 1.0032 - accuracy: 0.4706
Epoch 00008: val_accuracy did not improve from 0.55714
1/1 [==============================] - 5s 5s/step - loss: 1.0032 - accuracy: 0.4706 - val_loss: 0.9782 - val_accurac
y: 0.4857
Epoch 9/20
1/1 [==============================] - ETA: 0s - loss: 0.9997 - accuracy: 0.4740
Epoch 00009: val_accuracy did not improve from 0.55714
1/1 [==============================] - 5s 5s/step - loss: 0.9997 - accuracy: 0.4740 - val_loss: 0.9807 - val_accurac
y: 0.4857
Epoch 10/20
1/1 [==============================] - ETA: 0s - loss: 1.0039 - accuracy: 0.4706
Epoch 00010: val_accuracy did not improve from 0.55714
1/1 [==============================] - 5s 5s/step - loss: 1.0039 - accuracy: 0.4706 - val_loss: 0.9797 - val_accurac
y: 0.4857
Epoch 11/20
1/1 [==============================] - ETA: 0s - loss: 1.0027 - accuracy: 0.4706
Epoch 00011: val_accuracy did not improve from 0.55714
1/1 [==============================] - 5s 5s/step - loss: 1.0027 - accuracy: 0.4706 - val_loss: 0.9752 - val_accurac
y: 0.4857
Epoch 12/20
1/1 [==============================] - ETA: 0s - loss: 1.0036 - accuracy: 0.4706
Epoch 00012: val_accuracy did not improve from 0.55714
1/1 [==============================] - 5s 5s/step - loss: 1.0036 - accuracy: 0.4706 - val_loss: 0.9967 - val_accurac
y: 0.4857
Epoch 13/20
1/1 [==============================] - ETA: 0s - loss: 1.0053 - accuracy: 0.4706
Epoch 00013: val_accuracy did not improve from 0.55714
1/1 [==============================] - 6s 6s/step - loss: 1.0053 - accuracy: 0.4706 - val_loss: 0.9843 - val_accurac
y: 0.4857
Epoch 14/20
1/1 [==============================] - ETA: 0s - loss: 1.0023 - accuracy: 0.4740
Epoch 00014: val_accuracy did not improve from 0.55714
1/1 [==============================] - 6s 6s/step - loss: 1.0023 - accuracy: 0.4740 - val_loss: 0.9847 - val_accurac
y: 0.4857
Epoch 15/20
1/1 [==============================] - ETA: 0s - loss: 1.0029 - accuracy: 0.4671
Epoch 00015: val_accuracy did not improve from 0.55714
1/1 [==============================] - 5s 5s/step - loss: 1.0029 - accuracy: 0.4671 - val_loss: 0.9825 - val_accurac
y: 0.4857
Epoch 16/20
1/1 [==============================] - ETA: 0s - loss: 1.0021 - accuracy: 0.4671
Epoch 00016: val_accuracy did not improve from 0.55714
1/1 [==============================] - 6s 6s/step - loss: 1.0021 - accuracy: 0.4671 - val_loss: 0.9802 - val_accurac
y: 0.4857
Epoch 17/20
1/1 [==============================] - ETA: 0s - loss: 0.9974 - accuracy: 0.4671
Epoch 00017: val_accuracy did not improve from 0.55714
1/1 [==============================] - 6s 6s/step - loss: 0.9974 - accuracy: 0.4671 - val_loss: 0.9854 - val_accurac
y: 0.4857
Epoch 18/20
1/1 [==============================] - ETA: 0s - loss: 1.0022 - accuracy: 0.4740
Epoch 00018: val_accuracy did not improve from 0.55714
1/1 [==============================] - 5s 5s/step - loss: 1.0022 - accuracy: 0.4740 - val_loss: 0.9856 - val_accurac
y: 0.4857
Epoch 19/20
1/1 [==============================] - ETA: 0s - loss: 1.0006 - accuracy: 0.4706
Epoch 00019: val_accuracy did not improve from 0.55714
1/1 [==============================] - 6s 6s/step - loss: 1.0006 - accuracy: 0.4706 - val_loss: 0.9865 - val_accurac
y: 0.4857
Epoch 20/20
1/1 [==============================] - ETA: 0s - loss: 1.0031 - accuracy: 0.4706
Epoch 00020: val_accuracy did not improve from 0.55714
1/1 [==============================] - 6s 6s/step - loss: 1.0031 - accuracy: 0.4706 - val_loss: 0.9740 - val_accurac
y: 0.4857
```

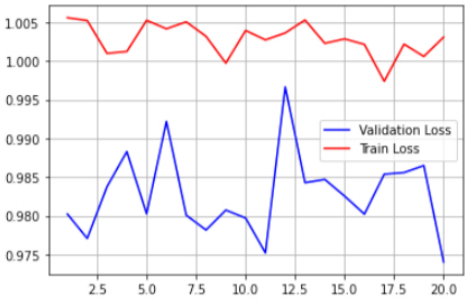
Entrée [43]:

```
fig, ax = plot.subplots(1,1)
ax.set_xlabel = "epoch"
ax.set_ylabel = "Accuracy"
```

```
x= list(range(1, 20+1))
vy = history.history['val_accuracy']
ty = history.history['accuracy']
plot_dynamic(x, vy, ty, ax)

#Loss
fig, ax = plot.subplots(1,1)
ax.set_xlabel = "epoch"
ax.set_ylabel = "Categorical crossentropy Loss"
x= list(range(1, 20+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_dynamic(x, vy, ty, ax)
```





Entrée [45]:
```
train_data.batch_size = train_data.n
(x, y) = train_data.next()
preds = cnn_model.predict(x)
train_preds_label = np.argmax(preds, axis=1)
train_true_label = np.argmax(y, axis=1)

validation_data.batch_size = validation_data.n
(x, y) = validation_data.next()
preds = cnn_model.predict(x)
validation_preds_label = np.argmax(preds, axis=1)
validation_true_label = np.argmax(y, axis=1)

print("Training Set Confusion Matrix : ")
print(CM(train_true_label, train_preds_label))

print("Validation Set Confusion Matrix : ")
print(CM(validation_true_label, validation_preds_label))
```

```
Training Set Confusion Matrix :
[[  1 109   0]
 [  1 135   0]
 [  0  43   0]]
Validation Set Confusion Matrix :
[[ 0 27  0]
 [ 0 34  0]
 [ 1  8  0]]
```

After trying many optimizers I can"t manage to make validation not be in stagnation on learning. 55% is best I managed to get.

5. Conclusion

To conclude the results are not really good. The best model is doing slightly better than random. Each are each model results:

- Linear model :
- Multi Layer Perceptron : **55%**
- Support Vector Machine : **53%**
- Convolutional Neural network : **55%**

The applicative problem had many difficulties:

- Each car brand has very different features depending on year, model, finition, color etc
- Computing is long for such images
- The angle of the photos were limites
- A better approach would have been to use a object detection Algorithm such as **YOLO** and then build a model for logo recognition which would have way more accurate. The dataset would have been easier to build and it would have been easier to add new brands which is the goal of a such application.