

University Of Hyderabad



Post Graduation Diploma Diploma Thesis

Project-Cdiscount's Image Classification Challenge

By

Sarthak Kaushik

(Sarthak.kaushik.17@gmail.com)

A thesis submitted in fulfilment for the PGDM in AI/ML
Under Guidance of
Manoj Sir & Srikanth Varma



Table Of Content

Declaration of Authorship	3
Abstract	4
Introduction	5
Literature Survey & Data Acquisition	6
1. Problem Definition:	6
2. Dataset:	6
3. Key metric (KPI) to optimise:	9
4. Real world challenges and constraints:	10
5. How are similar problems solved in literature (technical articles & blogs,forums, research papers)	11
EDA and Feature Extraction	13
1.Exploring Categories_name.csv file:	13
2. Exploring Train_example BSON File:	15
3. Exploring Train.BSON File:	16
4. Exploring Test.BSON File:	21
Modelling and Error Analysis	22
Base-Line model and metrics:Model-1-ResNet50	22
Model 2 -Xception	23
Model 3- EfficientNetB7	24
Model 3- EfficientNetB7(Fine- Tuned):	25
Advance Modelling and Feature Engineering	28
SOTA- EfficientNetB7(Without Fine Tuning):	28
2. SOTA- EfficientNetB7(With Fine Tuning):	31
Results	33
Deployment and Productionisation	35
Downloading the final model	35
Deploying model as a webapp using STREAMLIT	35
Deploying Streamlit App on web using Heruko	37
CONCLUSION	37
References	37

Declaration of Authorship

I, Sarthak Kaushik, declare that this thesis titled, 'Cdiscount Image Classification' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a diploma degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Abstract

This project is a kaggle problem challenge and has been there in kaggle from 2018.

The reason for taking this project is to help building our understanding on implementing various deep learning approaches that we have learned during this whole course and implement it on a live project to get a good hands on idea.

The write will help us to understand how we proceed with project from data preprocessing , model building to deployment as an webapp.

This project is a multi class classification on a very large data set (Train- 58.2GB and Test - 14.5 GB) with more than 5k classes to classify.

This project has been undertaken with the limited hardware resource and trying to achieve best accuracy possible.

Introduction

[Cdiscount.com](#) generated nearly 3 billion euros last year, making it France's largest non-food e-commerce company. While the company already sells everything from TVs to trampolines, the list of products is still rapidly growing. By the end of this year, Cdiscount.com will have over 30 million products up for sale. This is up from 10 million products only 2 years ago. Ensuring that so many products are well classified is a challenging task.

Currently, Cdiscount.com applies machine learning algorithms to the text description of the products in order to automatically predict their category. As these methods now seem close to their maximum potential, Cdiscount.com believes that the next quantitative improvement will be driven by the application of data science techniques to images.

In this challenge you will be building a model that automatically classifies the products based on their images. As a quick tour of Cdiscount.com's website can confirm, one product can have one or several images. The data set Cdiscount.com is making available is unique and characterised by superlative numbers in several ways:

- Almost 9 million products: half of the current catalogue
- More than 15 million images at 180x180 resolution
- More than 5000 categories: yes this is quite an extreme multi-class classification!

Literature Survey & Data Acquisition

1. Problem Definition:

Cdiscount is the France one of the largest ecommerce companies and have one of the largest databases of its product and images in their repositories. Now using this data they want to identify- given the image from their dataset identify the category of the product. This make the problem perfect example of Multiclass classification problem using image data as an input.

Thus the objective is to evaluate based on the categorization accuracy of the predictions that our model will make for each of the products.

What makes this problem even more interesting is the huge dataset (Train- 58.2GB and Test - 14.5 GB) cdiscount have provided us to build the model on. To work on such a huge dataset **efficiently** will give you more industry relevant experience in the world of BIG DATA.

Solving this problem will give you a much better understanding of how real world e-commerce or for that matter any other company manages to deal with huge data and how the classification algorithm is deployed in the real world without human intervention.

2. Dataset:

The dataset has been taken from Cdiscount Kaggle competition. Cdiscount provides the mongodb db BSON file. These files are as follows.

- a. **Train.bson**- This file contains a list of 7 million dictionaries and each dictionary represents a product. Each dictionary has :- `_id` field (product ID), `category_id`(category ID of the product) and `imgs`(list of stored 1-4 pictures of the product). The size of **Train.BSON file is approx 60GB**

```

for item in TRAIN_DB:
    break
print(type(item), list(item.keys()))
print(item['_id'], len(item['imgs']), item['category_id'],)

```

```

<class 'dict'>['_id', 'imgs', 'category_id']
0 1 1000010653

```

Nb of images	#1	#2	#3	#4
Nb of products	4,369,441	1,128,588	542,792	1,029,075

Table 3: Number of products having exactly 1, 2, 3 or 4 associated images.

- b. Test.bson** - This file contains a list of 1.7 million dictionaries and each dictionary represents a product. This data has all the fields as Train.bson except the 'category_id' (which we need to predict). The size of Test..BSON file is approx 15GB

Category level	Cat I	Cat II	Cat III
Nb of categories	49	483	5270

Table 2: Number of values taken by the 3 levels of the categorization tree.

category	category_level1	category_level2	category_level3
1000018296	79640	MUSIQUE	CD
1000011423	71116	INFORMATIQUE	IMPRESSION - SCANNER
1000011427	69784	INFORMATIQUE	IMPRESSION - SCANNER
1000014202	65642	LIBRAIRIE	LITTERATURE
1000015309	65435	LIBRAIRIE	AUTRES LIVRES
1000004085	61942	INFORMATIQUE	CONNECTIQUE - ALIMENTATION
1000010653	61688	TELEPHONIE - GPS	ACCESSOIRE TELEPHONE
1000018290	60332	MUSIQUE	CD
1000018294	57748	MUSIQUE	CD
1000008094	56192	INFORMATIQUE	COMPOSANT - PIECE DETACHEE
1000004079	55656	INFORMATIQUE	CONNECTIQUE - ALIMENTATION
1000005509	51332	AUTO - MOTO	CONFORT CONDUCTEUR ET PASSAGER
1000015912	49780	INFORMATIQUE	COMPOSANT - PIECE DETACHEE
1000010636	48488	TELEPHONIE - GPS	ACCESSOIRE TELEPHONE
1000011349	47691	TELEPHONIE - GPS	ACCESSOIRE TELEPHONE

Challenges: Since the dataset that we need to process is huge, there are a lot of challenges that were coming while processing the data.

- **Cloud Storage-** Since we would be using google colab for our project so its a prerequisite to store our data in google drive. And since google drive provide only 15GB data by default, we need to further purchase the space of 200GB to load are data properly and to even store some the extra data generated from preprocessing step in the google drive only.
- **Google Colab-** Due to the large dataset the resource required to process data and train the data is HUGE! Even though google colab provides decent hardware for the start, but to deal with this kind of data we might need to purchase colab Pro to improve the time to process this data, if required
- **Bson file preprocessing-** Because of huge data, we need to employ multithreading techniques to process the bson file, and need to understand the new library 'Import bson' to read the bson data.

Tools/Library required:

```
import os, sys, math, io
import numpy as np
import pandas as pd
import multiprocessing as mp
import bson
import struct
import matplotlib.pyplot as plt
import keras
from keras.preprocessing.image import load_img, img_to_array
import tensorflow as tf
from tensorflow import keras
from collections import defaultdict
from tqdm import *
```

3. Key metric (KPI) to optimise:

Since this is the task of multiclass classification, we can use the following metrics:-

Confusion Matrix

		Actual	
		Positives(1)	Negatives(0)
Predicted	Positives(1)	TP	FP
	Negatives(0)	FN	TN

Confusion Matrix

TP- True Positive
TN- True Negative
FP- False Positive
FN- False Negative

a. Accuracy:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

b. Precision

$$Precision = \frac{TP}{TP+FP}$$

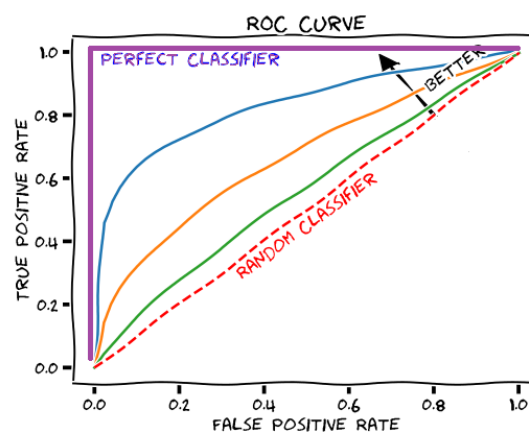
c. Recall

$$Recall = \frac{TP}{TP+FN}$$

d. F1-score

$$F1 - score = \frac{2 * Recall * Precision}{Recall + Precision}$$

e. ROC AUC curve (using one class versus the rest methodology)



Failure of these metrics:

- 1) ROC AUC will fail if we have very high imbalances in our classes. In such cases we should use F1 score as a metric over the ROC AUC curve.
- 2) Accuracy can give false assumptions regarding the classifier performance. In such cases its better to use Precision and Recall

Note: These metrics can be used in all the classification tasks.

4. Real world challenges and constraints:

1. Few of the challenges in this project is to scale the training process.

- a. HardDisk I/O speeds are very slow and this will act as a bottleneck for feeding the data into our CNN even with different batch sizes.
 - b. There will be large numbers of parameters/weights to be learned and to do it efficiently will be a challenge.
 - c. There will be large numbers of parameters/weights to be learned and to do it efficiently will be a challenge.
2. Solution to manage these challenge to an extent:
- a. Need to subscribe more powerful hardware like - Google Colab Pro or genesis cloud systems
 - b. Since we can't train the data in one shot , it would be better to train the models first on 20-30% data to get the idea how the models are performing and then taking the informed decision we can increase the training input into the CNN.

5. How are similar problems solved in literature (technical articles & blogs,forums, research papers)

How are similar problems solved in literature (technical articles & blogs, forums, research papers)

We can employ following two approaches:

- Strategy 1 - Pre-trained Models.
 - Idea- Train a model from existing pre-trained weights and prune the model based on the given dataset to expedite the overall training process.
 - Models to implement -
 - ResNet50
 - ResNet101
 - InceptionV3

■ EfficientNet architectures

- Strategy 2:- An Ensemble of Pre-Trained Models :
 - Idea : Instead of trying out one model at a time we can combine the predictive results across several different models to increase the predictive power of our Deep Learning models. We can achieve this by ultimately introducing an ensemble.
 - Since the model will be trained with different hyperparameters on different subset of data, hence we may see some improvement in the performance.
 - We can utilize an averaging strategy to utilize the average probability across two of the models for the inference process.

EDA and Feature Extraction

Since we are working on the image dataset for our image classification task with hardly any columns available (apart from categories and product id) there is no much to explore in this data. But we will still try to dig deep with whatever data available we have. We will try to explore the data inside all the data file that have been provided to us in the following manner.

1. Categories_name.csv
2. Train.bson
3. Test.bson

1.Exploring Categories_name.csv file:

This file contains the hierarchy of product classification. For each category_id we have a hierarchy of three different levels: Level-1, Level-2 and Level 3 . PFB below the images of the data representation for the category_id.

```
[ ] # categories_df = pd.read_csv(train_bson_path, index_col='category_id')
categories_df = pd.read_csv(train_bson_path)
```



```
[ ] categories_df.head()
```

	category_id	category_level1	category_level2	category_level3
0	1000021794	ABONNEMENT / SERVICES	CARTE PREPAYEE	CARTE PREPAYEE MULTIMEDIA
1	1000012764	AMENAGEMENT URBAIN - VOIRIE	AMENAGEMENT URBAIN	ABRI FUMEUR
2	1000012776	AMENAGEMENT URBAIN - VOIRIE	AMENAGEMENT URBAIN	ABRI VELO - ABRI MOTO
3	1000012768	AMENAGEMENT URBAIN - VOIRIE	AMENAGEMENT URBAIN	FONTAINE A EAU
4	1000012755	AMENAGEMENT URBAIN - VOIRIE	SIGNALETIQUE	PANNEAU D'INFORMATION EXTERIEUR

Following are the key observation from the categories.csv file

A) Unique Categories:

- Unique category_id: 5270
- Unique categories_level1 values: 49
- Unique categories_level2 values: 483
- Unique categories_level3 values: 5263

Observation: the category level 3 is approximately equal to unique category_id.

This means we can assume category_id as one to one mapping with category_level3 i.e. for each category_id we can have its corresponding string name. Still we see that there are 7 values from categories that are not getting unique. This means those 7 values are appearing more than once in the data

Below are the list of those 7 value.

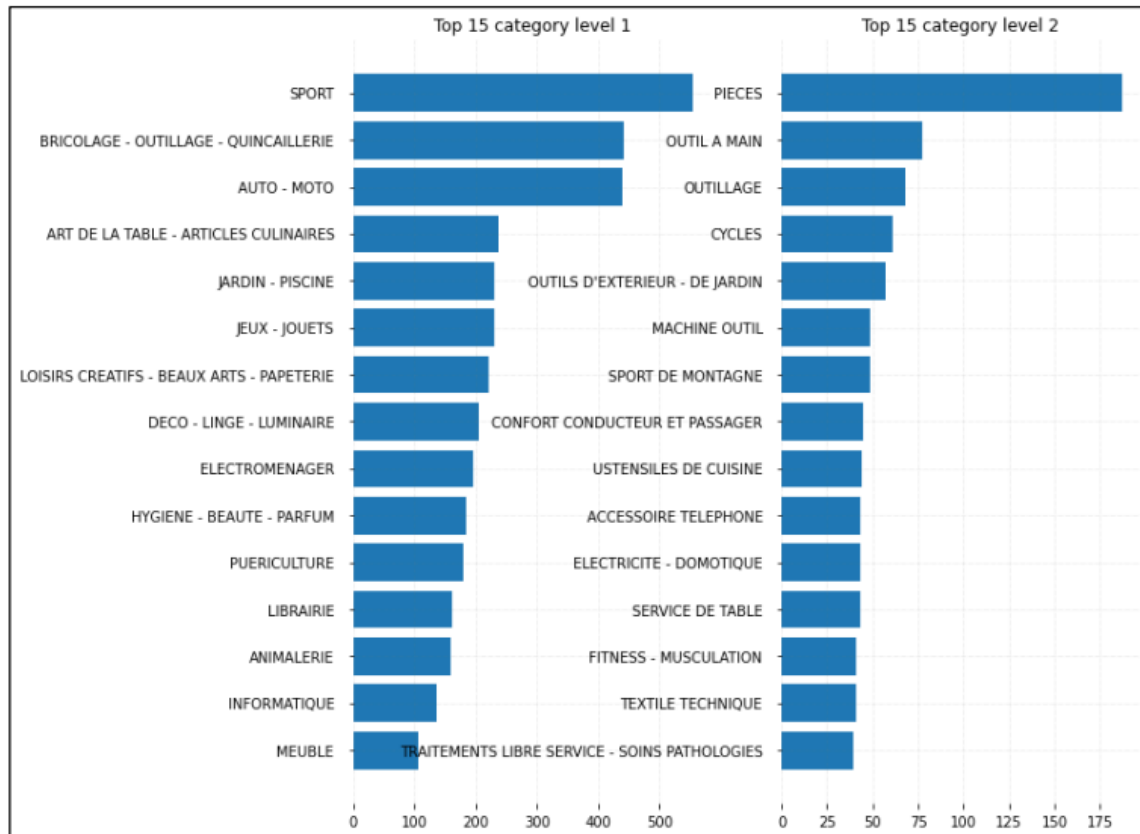
```
#let's dig deep and see do we observe any observation between category id and category 3

gb_obj = categories_df.groupby('category_level3')
gb_cnt = gb_obj.count()
# print(cnt)
print('No of rows with one-one mapping between category_id and Category 3 - ',len(gb_cnt[gb_cnt['category_id'] ==1]))
print('No of rows with more than one category_id for a given Category 3 - ',len(gb_cnt[gb_cnt['category_id'] >1]))
print(gb_cnt[gb_cnt['category_id'] >1])
```

No of rows with one-one mapping between category_id and Category 3 - 5256
No of rows with more than one category_id for a given Category 3 - 7

category_level3	category_id	...	category_level2
CONFORT URINAIRE	2	...	2
FONTAINE A EAU	2	...	2
FUSIBLE	2	...	2
GUIDON	2	...	2
PELUCHE	2	...	2
PROTEGE ECRAN - FILM DE PROTECTION	2	...	2
VOITURE	2	...	2

B) Top 15 values of Category Level 1 and Category level 2



2. Exploring Train_example BSON File:

This file is smaller version of the large train.bson file. This file will help us to load our data into the memory easily and look at the some of the attributes of the file.

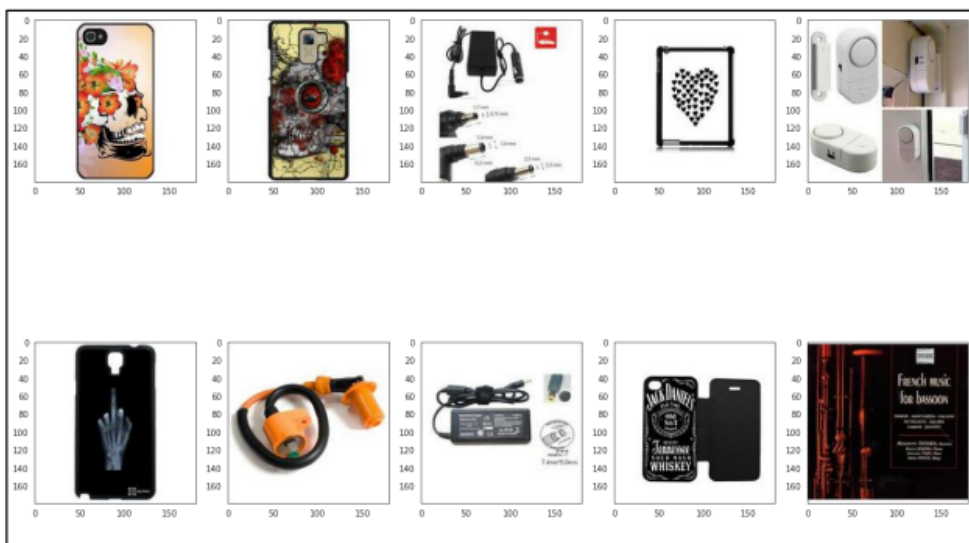
This file is also the list of dictionaries and each dictionary represents a product. Each dictionary has :- `_id` field (product ID), `category_id` (category ID of the product) and `imgs` (list of stored 1-4 pictures of the product)

The data structure inside the file looks like as show in below image.

```
[ ] df_example.head()
```

	_id	imgs	category_id
0	0	<PIL.JpegImagePlugin.JpegImageFile image mode=...	1000010653
1	1	<PIL.JpegImagePlugin.JpegImageFile image mode=...	1000010653
2	2	<PIL.JpegImagePlugin.JpegImageFile image mode=...	1000004079
3	3	<PIL.JpegImagePlugin.JpegImageFile image mode=...	1000004141
4	4	<PIL.JpegImagePlugin.JpegImageFile image mode=...	1000015539

Few of the images are loaded and they look as follows.



After seeing the train_example.bson file we get the fairly good idea how are data would be structured inside the train.bson file.

3. Exploring Train.BSON File:

This file contains a list of 7 million dictionaries and each dictionary represents a product. Each dictionary has :- _id field (product ID), category_id(category ID of the product) and imgs(list of stored 1-4 pictures of the product). The size of Train.BSON file is approx 60GB.

Note:- we have already seen some of the raw images in train_example.bson file.

Now since the file size large, therefore for our EDA purpose we won't be loading the image in the memory while reading our file. Instead, we would create a new variable

to capture number of images for a particular product. This will help us to give much better insight , as this will be a numeric value.

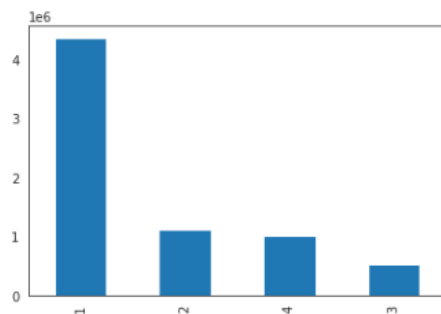
A) Number of Image per product:

of product with 1 image – 4,369,441 ~4.3million

of product with 2 image – 1,128,588 ~1.1 million

of product with 4 image – 1,029,075 ~ 1million

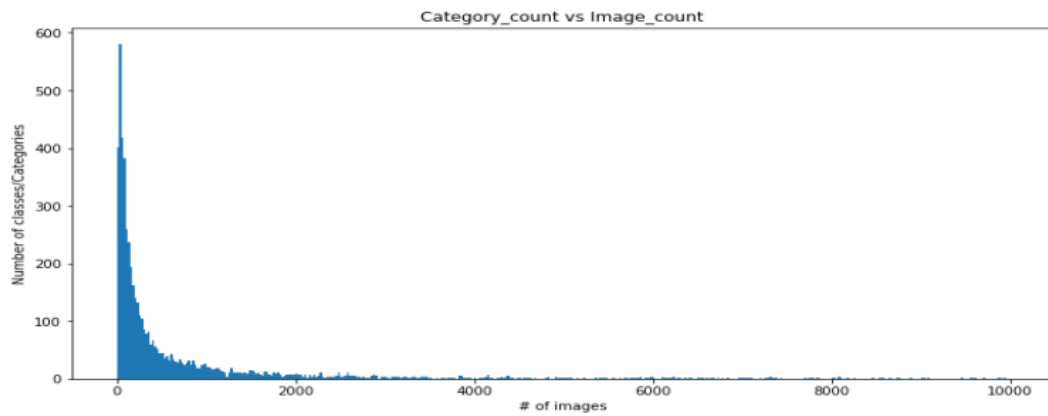
of product with 3 image – 542,792 ~ 500k



So this means we have more products which has only 1 images and going by this data looks quite skewed!

B) Category count vs Image Count

We can also plot relationship between numbers of categories that are presented by the total number of images: i.e. 10 categories are presented in the training dataset by 100 images. We will observe how much the data is skewed



C) Top 15 most common category joining train and category file.

```
[31] temp=df['category_id'].value_counts().to_frame().head(15).reset_index()
temp.columns=['category_id','counts']
# temp.head()
# temp['category_id'].to_list()
categories_df[categories_df['category_id'].isin(temp['category_id'].to_list())]
```

	category_id	category_level1	category_level2	category_level3
515	1000005509	AUTO - MOTO	CONFORT CONDUCTEUR ET PASSAGER	PERSONNALISATION VEHICULE - DECORATION VEHICULE
2501	1000015912	INFORMATIQUE	COMPOSANT - PIECE DETACHEE	CLAVIER (PIECE DETACHEE)
2503	1000008094	INFORMATIQUE	COMPOSANT - PIECE DETACHEE	DALLE D'ECRAN
2522	1000004085	INFORMATIQUE	CONNECTIQUE - ALIMENTATION	BATTERIE D'ALIMENTATION INFORMATIQUE
2530	1000004079	INFORMATIQUE	CONNECTIQUE - ALIMENTATION	CHARGEUR - ADAPTATEUR SECTEUR - ALLUME CIGARE ...
2548	1000011427	INFORMATIQUE	IMPRESSION - SCANNER	CARTOUCHE IMPRIMANTE
2560	1000011423	INFORMATIQUE	IMPRESSION - SCANNER	TONER - RECUPERATEUR DE TONER
3253	1000015309	LIBRAIRIE	AUTRES LIVRES	AUTRES LIVRES
3335	1000014202	LIBRAIRIE	LITTERATURE	LITTERATURE FRANCAISE
3956	1000018290	MUSIQUE	CD	CD MUSIQUE CLASSIQUE
3958	1000018294	MUSIQUE	CD	CD MUSIQUE DU MONDE
3959	1000018296	MUSIQUE	CD	CD POP ROCK - CD ROCK INDE
5043	1000010635	TELEPHONIE - GPS	ACCESSOIRE TELEPHONE	BATTERIE TELEPHONE
5055	1000010653	TELEPHONIE - GPS	ACCESSOIRE TELEPHONE	COQUE TELEPHONE - BUMPER TELEPHONE
5075	1000011349	TELEPHONIE - GPS	ACCESSOIRE TELEPHONE	PACK ACCESSOIRES

Observation – Top categories that product fall into are AUTO , COMPUTER , LIBRARY , TELEPHONY , DECO

D) Top 15 least common category joining train and category file.

```
# 'Let's see the least 15 least common category from the category table'
temp_df['category_id'].value_counts().to_frame().tail(15).reset_index()
temp.columns=['category_id','counts']
categories_df[categories_df['category_id'].isin(temp['category_id'].to_list())]
```

category_id	category_level1	category_level2	category_level3
981	1000022465	BATEAU MOTEUR - VOILIER	ELECTRICITE
1535	1000015609	CHAUSSURES - ACCESSOIRES	ACCESSOIRES CHAUSSURES
2241	1000000896	EPICERIE	CONSERVE DE LEGUME
3452	1000014467	LOISIRS CREATIFS - BEAUX ARTS - PAPETERIE	COLLECTION - PHILATELIE - CARTOPHILIE - NUMISM...
3667	1000015046	MATERIEL DE BUREAU	MATERIEL PEDAGOGIQUE
3711	1000011519	MATERIEL MEDICAL	ACUPUNCTURE - MEDECINES PARALLELES
3785	1000011955	MATERIEL MEDICAL	SOIN
3846	1000019484	MEUBLE	ACCESSOIRE DE MEUBLE
4154	1000019608	PHOTO - OPTIQUE	PIECES DETACHEES PHOTO - OPTIQUE
4179	1000010893	PHOTO - OPTIQUE	VISIONNAGE PHOTO
4395	1000007760	PUERICULTURE	TOILETTE BEBE
4451	1000019804	SPORT	BASEBALL
4535	1000007168	SPORT	CYCLES
5215	1000022325	TV - VIDEO - SON	LECTEUR MUSIQUE
5228	1000020847	TV - VIDEO - SON	PROTECTION - ENTRETIEN

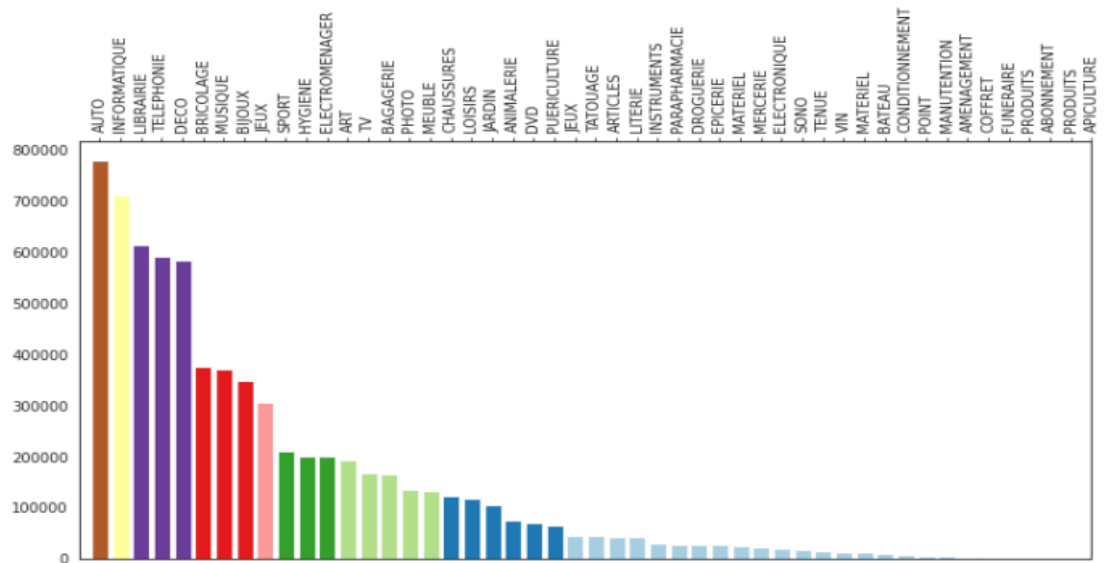
E) Lets see the Category Level1 in train data.

Most Common Categories

```
#Lets see the Category Level1 in train data

catlvl1 = train['category_level1'].value_counts()
# cats.head(15)
print ('No. of Category level1 :',len(catlvl1))
print(catlvl1.head())
```

```
No. of Category level1 : 49
AUTO - MOTO          779158
INFORMATIQUE         711866
LIBRAIRIE            613354
TELEPHONIE - GPS     591516
DECO - LINGE - LUMINAIRE 584467
Name: category_level1, dtype: int64
```



Observation – Top categories that product fall into are AUTO , COMPUTER , LIBRARY , TELEPHONY , DECO

Building the word cloud for category level1 for the train data.



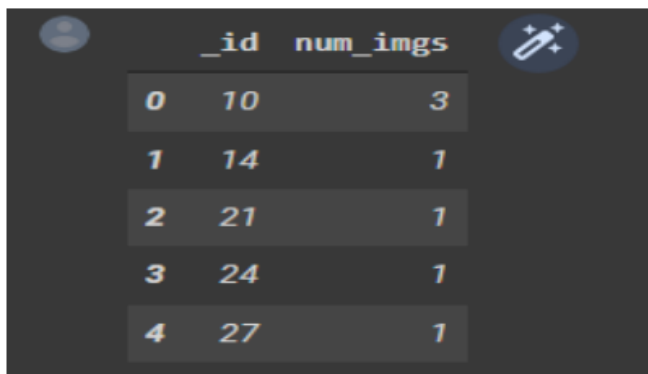
Building the word cloud for category level 2 for the train data.



4. Exploring Test.BSON File:

This file contains a list of 1.7 million dictionaries and each dictionary represents a product. Each dictionary has :- `_id` field (product ID) and `imgs` (list of stored 1-4 pictures of the product). This data has all the field as `Train.bson` except the `'category_id'` (which we need to predict). The size of `Test..BSON` file is approx 15GB.

We don't have much to explore in test data , we can only look at the data structure of this bson file and then proceed with model building. Since the file size large, therefore we won't be loading the image in the memory while reading our file. Instead, we would create a new variable to capture number of images for a particular product.



A screenshot of a Jupyter Notebook interface showing a table with two columns: `_id` and `num_imgs`. The table contains five rows of data. The first row has `0` for `_id` and `3` for `num_imgs`. The second row has `1` for `_id` and `1` for `num_imgs`. The third row has `2` for `_id` and `1` for `num_imgs`. The fourth row has `3` for `_id` and `1` for `num_imgs`. The fifth row has `4` for `_id` and `1` for `num_imgs`. The table is displayed in a dark-themed interface with a search icon on the left and a plus icon on the right.

	<code>_id</code>	<code>num_imgs</code>
0	10	3
1	14	1
2	21	1
3	24	1
4	27	1

Final objective - The goal of the competition is to predict `category_id` by image. We need to predict a number, e.g. 1000010653 by an image

Notebook link-

[Cdiscount-Webapp_deployment_heroku/Cdiscount_EDA_Phase2.ipynb at main · sarthakkaushik/Cdiscount-Webapp_deployment_heroku \(github.com\)](#)

Modelling and Error Analysis

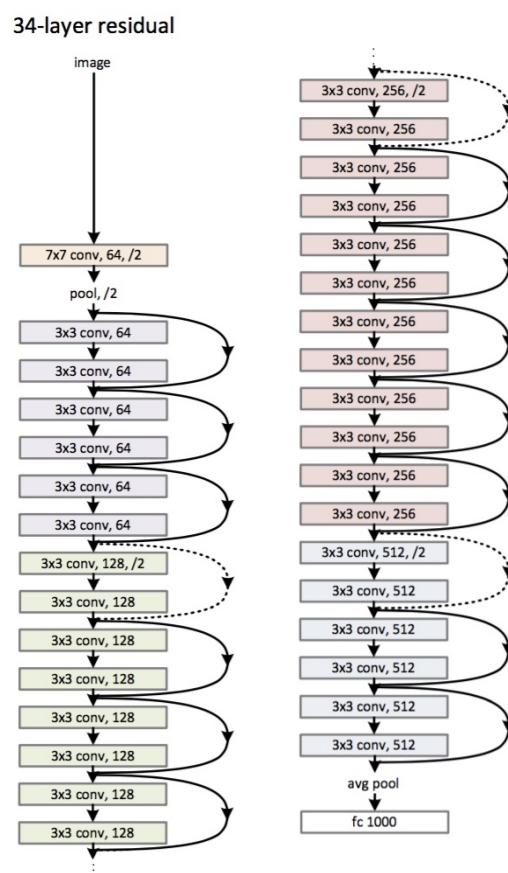
Perquisite:

Before going for model building the data was formatted into the right format and for this I have used some of the online kernels. I have built some of the helper functions to create loss and accuracy graphs.

Model CheckPoints Callbacks were extensively used for each model that were tested.

Note-Since the data was huge we employing transfer learning on 10% of the train data

1. Base-Line model and metrics:Model-1-ResNet50



- The base model we tried is Resnet50 without the top layer
- All the layers of the base model was non trainable.

- The base was connected with GlobalAveragePooling2D followed by the output Dense layer
- The metric that we use accuracy (though I should have used F1-score but I realised it later)
- Optimizer used was Adam with default learning Rate.
- The model summary with trainable parameters looks like below.

```
[ ] #Getting the summary of the model we created
model_2.summary()

Model: "model_2"

Layer (type)                 Output Shape              Param #
-----
input_layers (InputLayer)    [(None, 180, 180, 3)]    0
resnet50 (Functional)        (None, None, None, 2048) 23587712
GlobalAveragePooling2D_layers (GlobalAveragePooling2D) (None, 2048)             0
output_layers (Dense)        (None, 5270)             10798230
-----
Total params: 34,385,942
Trainable params: 10,798,230
Non-trainable params: 23,587,712
```

- To run 1 epoch it took more than 2 hours, hence we ran this model for only 3 epochs.
- Training and Validation accuracy is as follows.
 - Training Accuracy- 51.75%
 - Validation Accuracy- 44.13%

2. Model 2 -Xception

- The 2 nd model we tried is Xception without the top layer.
- All the layers of the base model were non-trainable.
- The base was connected with GlobalAveragePooling2D followed by the output Dense layer • The metric that we use accuracy (though I should have

used F1-score but I realised it later) • Optimizer used was Adam with default learning Rate.

- The model summary with trainable parameters looks like below.

Layer (type)	Output Shape	Param #
input_layers (InputLayer)	[(None, 180, 180, 3)]	0
xception (Functional)	(None, None, None, 2048)	20861480
GlobalAveragePooling2D_layers (GlobalAveragePooling2D)	(None, 2048)	0
output_layers (Dense)	(None, 5270)	10798230
Total params: 31,659,710		
Trainable params: 10,798,230		
Non-trainable params: 20,861,480		

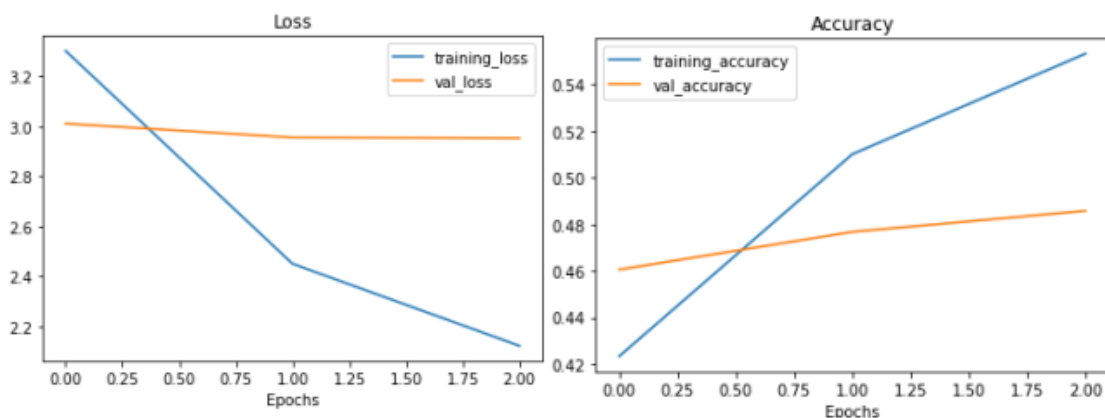
- **To run 1 epoch it took more than 95 minutes, hence we ran this model for only 3 epochs.**
- Training and Validation accuracy is as follows.
 - Training Accuracy- 15.93%
 - Validation Accuracy- 16.58%

3. Model 3- EfficientNetB7

- The 3rd model we tried is EfficientNetB7 without the top layer.
- All the layers of the base model was non trainable.
- The base was connected with GlobalAveragePooling2D followed by the output Dense layer • The metric that we use accuracy (though I should have used F1-score but I realized it later) • Optimizer used was Adam with default learning Rate.
- The model summary with trainable parameter looks like below

Layer (type)	Output Shape	Param #
input_layers (InputLayer)	[(None, 180, 180, 3)]	0
efficientnetb7 (Functional)	(None, None, None, 2560)	64097687
GlobalAveragePooling2D_layers (GlobalAveragePooling2D)	(None, 2560)	0
output_layers (Dense)	(None, 5270)	13496470
=====		
Total params: 77,594,157		
Trainable params: 13,496,470		
Non-trainable params: 64,097,687		

- To run 1 epoch it took more than 2 hours, hence we ran this model for only 3 epochs.
- Training and Validation accuracy is as follows-
 - Training Accuracy- 55.33%
 - Validation Accuracy- 48.58%



Note:- The result looks substantially well as compare to other models that we tried, but there is a problem of overfitting ,hence I tried to do fine tuning on this model.

4. Model 3- EfficientNetB7(Fine- Tuned):

The model was fine-tuned for the last 12 layer of EfficientNetB7 model.

- We have to unfreeze the top 12 layers and trained our model for another 3 epochs.
- This means all of the base model's layers except for the last 12 will remain frozen and untrainable. And the weights in the remaining unfrozen layers will be updated during training.
- In our case, we're using the exact same loss, optimizer and metrics as before, except this time the learning rate for our optimizer will be 10x smaller than before (0.0001 instead of Adam's default of 0.001)
- This means all of the base model's layers except for the last 12 will remain frozen and untrainable. And the weights in the remaining unfrozen layers will be updated during training.
- In our case, we're using the exact same loss, optimizer and metrics as before, except this time the learning rate for our optimizer will be 10x smaller than before (0.0001 instead of Adam's default of 0.001)
- The model summary with trainable parameters looks like below.

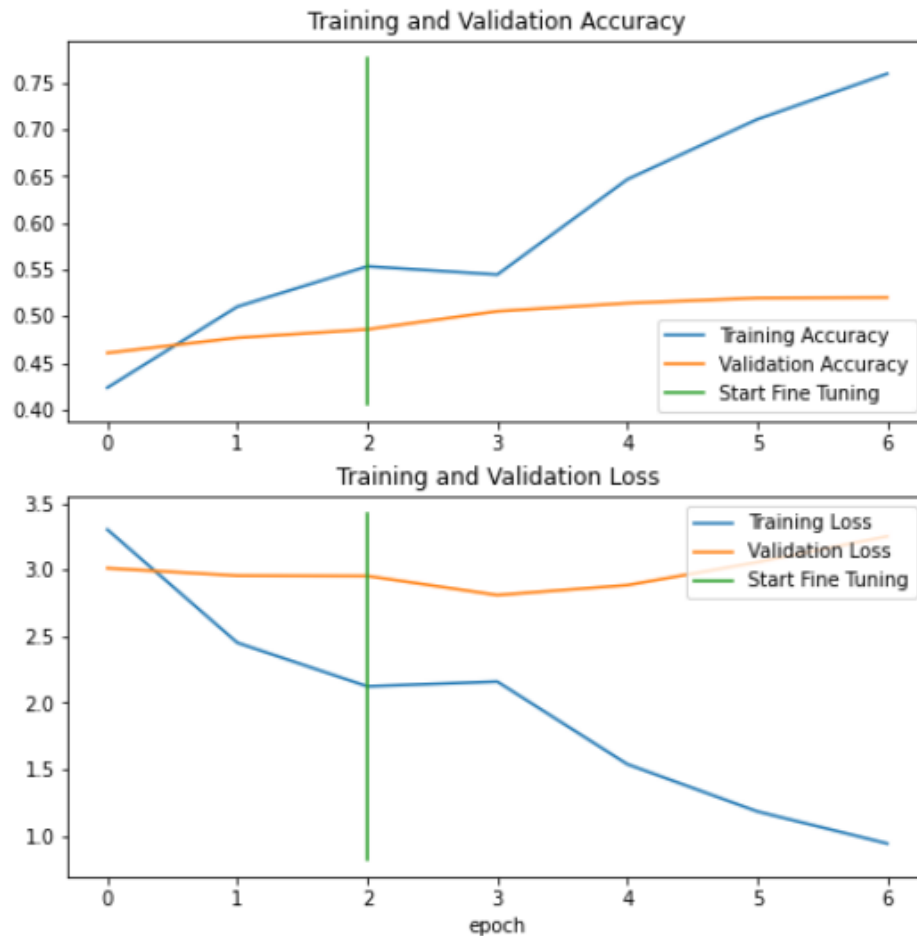
Model: "model"

Layer (type)	Output Shape	Param #
input_layers (InputLayer)	[(None, 180, 180, 3)]	0
efficientnetb7 (Functional)	(None, None, None, 2560)	64097687
GlobalAveragePooling2D_layers (GlobalAveragePooling2D)	(None, 2560)	0
output_layers (Dense)	(None, 5270)	13496470

=====

Total params: 77,594,157
 Trainable params: 18,831,670
 Non-trainable params: 58,762,487

- To run further 1 epoch it took near about 75min
- Training and Validation accuracy is as follows.
 - Training Accuracy- 75.9%
 - Validation Accuracy- 52%



Note:- Although the training accuracy has increased but validation score still remain low. Our model is overfitting the data hence in future course we need see how we further fine tune our model or use better learning rate so that our model can generalize well.

Notebook link- [Cdiscount-Webapp_deployment_heroku/Cdiscount_Phase-3.ipynb at main · sarthakkaushik/Cdiscount-Webapp_deployment_heroku \(github.com\)](#)

Advance Modelling and Feature Engineering

Perquisite:-

- 1) Before going for model building the data was formatted into the right format and for this I have used some of the online kernels.
- 2) I have built some of the helper functions to create loss and accuracy graphs.
- 3) Model CheckPoints Callbacks were extensively used for each model that were tested.
- 4) Learning rate scheduler Call backs were used to improve the model Accuracy.
- 5) Mixed Precession training was used on GPU to improve the training time.
- 6) Train.bson file was directly copied from Google drive to Colab environment, to further improve the training time
- 7) **50 % of the data was deployed for the training process.**
- 8) **All the training logs were captured using Weights and Biases.**

1. SOTA- EfficientNetB7(Without Fine Tuning):

- This model we tried is EfficientNetB7 without the top layer.
- All the layers of the base model was non trainable.
- The base was connected with GlobalAveragePooling2D followed by Dropout layer and then the output Dense layer
- The metric that we use accuracy, precession, recall and F1-Score
- Optimizer used was Adam with default learning Rate.
- Keras Learning rate scheduler was used .
- We deployed the model on 50% of the data
- This model ran for 4 epochs.
- The model summary with trainable parameters looks like below.

```

Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb7\_notop.h5
258080768/258076736 [=====] - 4s 0us/step
258088960/258076736 [=====] - 4s 0us/step
Model: "model"

Layer (type)                 Output Shape                 Param #
-----
input_layers (InputLayer)    [(None, 180, 180, 3)]       0
efficientnetb7 (Functional)  (None, None, None, 2560)    64097687
GlobalAveragePooling2D_lay  (None, 2560)                0
ers (GlobalAveragePooling2D)
dropout (Dropout)            (None, 2560)                0
output_layers_float_32 (Den  (None, 5270)                13496470
se)

Total params: 77,594,157
Trainable params: 13,496,470
Non-trainable params: 64,097,687

```

- To run 1 epoch it took more approx 6 hours and we ran this model for 19 epochs.(including Fine Tuning)
- Training and Validation metrics are as follows.

Training- ▪ Accuracy- 0.5111
 ▪ Precision- 0.763
 ▪ Recall- 0.4135
 ▪ F1-Score-53.56% o Validation-
 ▪ Accuracy- 0.5158 ▪ Precision- 0.772
 ▪ Recall- 0.4257
 ▪ F1-Score-54.82%

NOTE:-Before using this method , many other strategy were explored and experimented with , like different type callback learning rate scheduler like exponential and Step scheduler were used.

Below are the setting used

```

# Creating learning rate reduction callback
reduce_lr =
tf.keras.callbacks.ReduceLROnPlateau(monitor="val_accuracy",
                                     factor=0.2, #
multiply the learning rate by 0.2 (reduce by 5x)
                                     patience=2,
                                     verbose=1, #
print out when learning rate goes down
                                     min_lr=1e-7)

```

```

optimizer = tf.keras.optimizers.Adam(learning_rate =
base_learning_rate)
class LRLogger(tf.keras.callbacks.Callback):
    def __init__(self, optimizer):
        super(LRLogger, self).__init__()
        self.optimizer =optimizer

    def on_epoch_end(self,epoch,logs):
        lr = self.optimizer.learning_rate(epoch)
        wandb.log({"lr":lr}, commit = False)

earlystopping_callback = tf.keras.callbacks.EarlyStopping(
    monitor="val_accuracy",
    min_delta=0,
    patience=3,
    verbose=0,
    mode="max",
    baseline=None,
    restore_best_weights=False,
)

# Setup checkpoint path
checkpoint_path = '/gdrive/MyDrive/UOH Assignment
Dataset/cdiscount/CheckPoints/NEW_EfficientNetB7-50_percent_data_
Fine_Tune/checkpoint.ckpt'
#note: remember saving directly to Colab is temporary

# Create a ModelCheckpoint callback that saves the model's
weights only
checkpoint_callback =
tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,

save_weights_only=True, # set to False to save the entire model

save_best_only=False, # set to True to save only the best model
instead of a model every epoch

save_freq="epoch",# save every epoch

monitor="val_accuracy",

```

```
mode='max',  
  
verbose=1)  
  
callbacks = [ checkpoint_callback,WandbCallback()]
```

```
efficientB7_model.compile(  
    optimizer=optimizer,  
    loss="categorical_crossentropy",  
    metrics=["accuracy", precision_m, recall_m, f1_m])
```

```
Hist_1= efficientB7_model.fit(  
    train_gen, epochs=epochs, callbacks=callbacks,  
    validation_data=val_gen,  
    )
```

2. SOTA- EfficientNetB7(With Fine Tuning):

The model is fine-tuned for the last 5 layer of EfficientNetB7 model.

We have to unfreeze the top 5 layers and trained our model for another 13 epochs.

- This means all of the base model's layers except for the last 5 will remain frozen and untrainable. And the weights in the remaining unfrozen layers will be updated during training.
- The metric that we use accuracy, precession, recall and F1-Score
- Optimizer used was Adam with default learning Rate.
- Keras Learning rate scheduler was used .
- We deployed the model on 50% of the data
- The model summary with trainable parameters looks like below.

```
efficientB7_model.summary()

Model: "model"

Layer (type)                 Output Shape              Param #
-----
input_layers (InputLayer)    [(None, 180, 180, 3)]    0

efficientnetb7 (Functional)   (None, None, None, 2560) 64097687

GlobalAveragePooling2D_layers (GlobalAveragePooling2D) (None, 2560) 0

dropout (Dropout)            (None, 2560)             0

output_layers_float_32 (Dense) (None, 5270)             13496470

Total params: 77,594,157
Trainable params: 13,496,470
Non-trainable params: 64,097,687
```

To run further 1 epoch it took near about 4.5 hrs

• Training and Validation metrics up to 18 epochs are as follows.

o Training-

- Accuracy- 0.58
- Precision- 0.8334
- Recall- 0.4618
- F1-Score-59.38

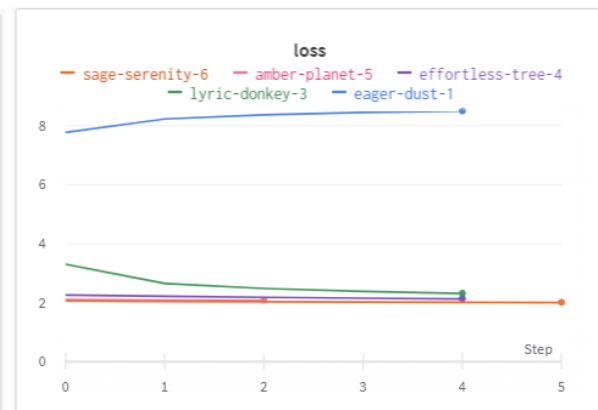
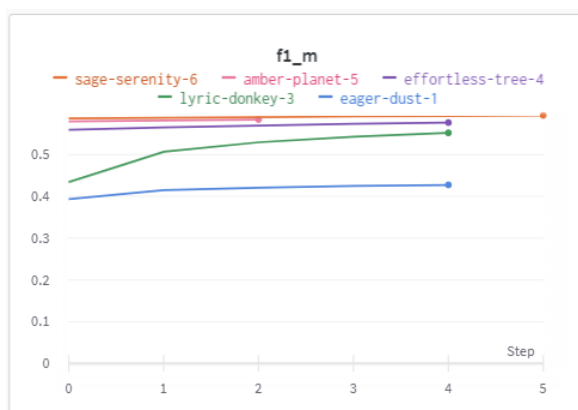
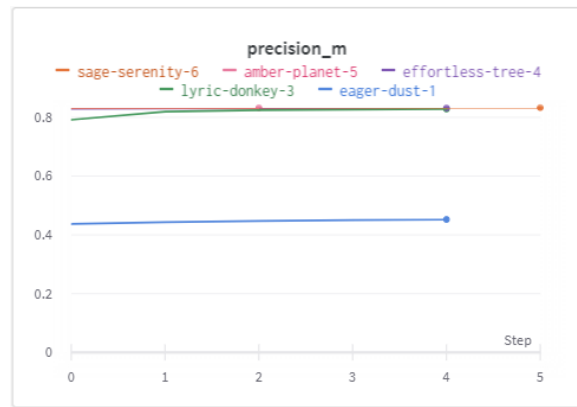
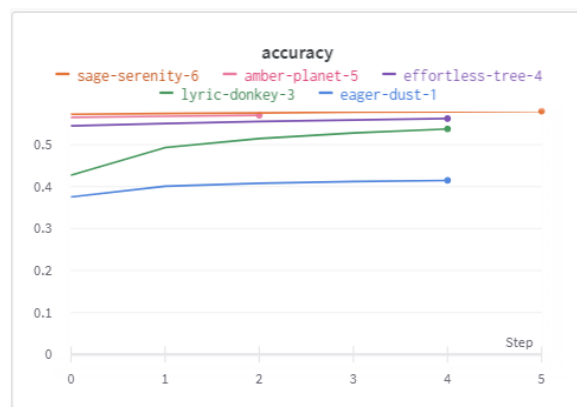
o Validation-

- Accuracy- 0.5615
- Precision- 0.8400
- Recall- 0.4570
- F1-Score-58.57%

Results

Below are the results of the performance metrics of the model training that were logged in weights & biases.

Please note during the training the process there were multiple disconnect from google colab server hence you will see multiple lines on each metric because once google colab get disconnected you need to rerun the code again from the last checkpoint but the weights & biases creates a new instance for it and map on the same project metric.



You can see the complete log in weights and biases using following link- [Cdiscount Performance Report – Weights & Biases \(wandb.ai\)](#)

Notebook link- [Cdiscount-Webapp_deployment_heroku/Cdiscount_Phase-4.ipynb at main · sarthakkaushik/Cdiscount-Webapp_deployment_heroku \(github.com\)](#)

Deployment and Productionisation

1. Downloading the final model

Since we have trained our model for more than 4 days we were able to achieve F1-score of 60 % on our validation data. This score could be improved if we would have trained on complete data for more number of epochs, but due to crunch hardware resource and time the model score is good enough to let us know how we can employ transfer learning on SOTA model to achieve our task.

For our deployment purpose we would be downloading our model with F1-score near about ~59% using below command

```
#####  
#Saving Model  
tf.keras.models.save_model(efficientB7_model,'Cdiscount.hdf5')  
  
#####
```

2. Deploying model as a webapp using STREAMLIT

After saving the model into your local machine, you can create a new python script and write the code to deploy the model using streamlit.

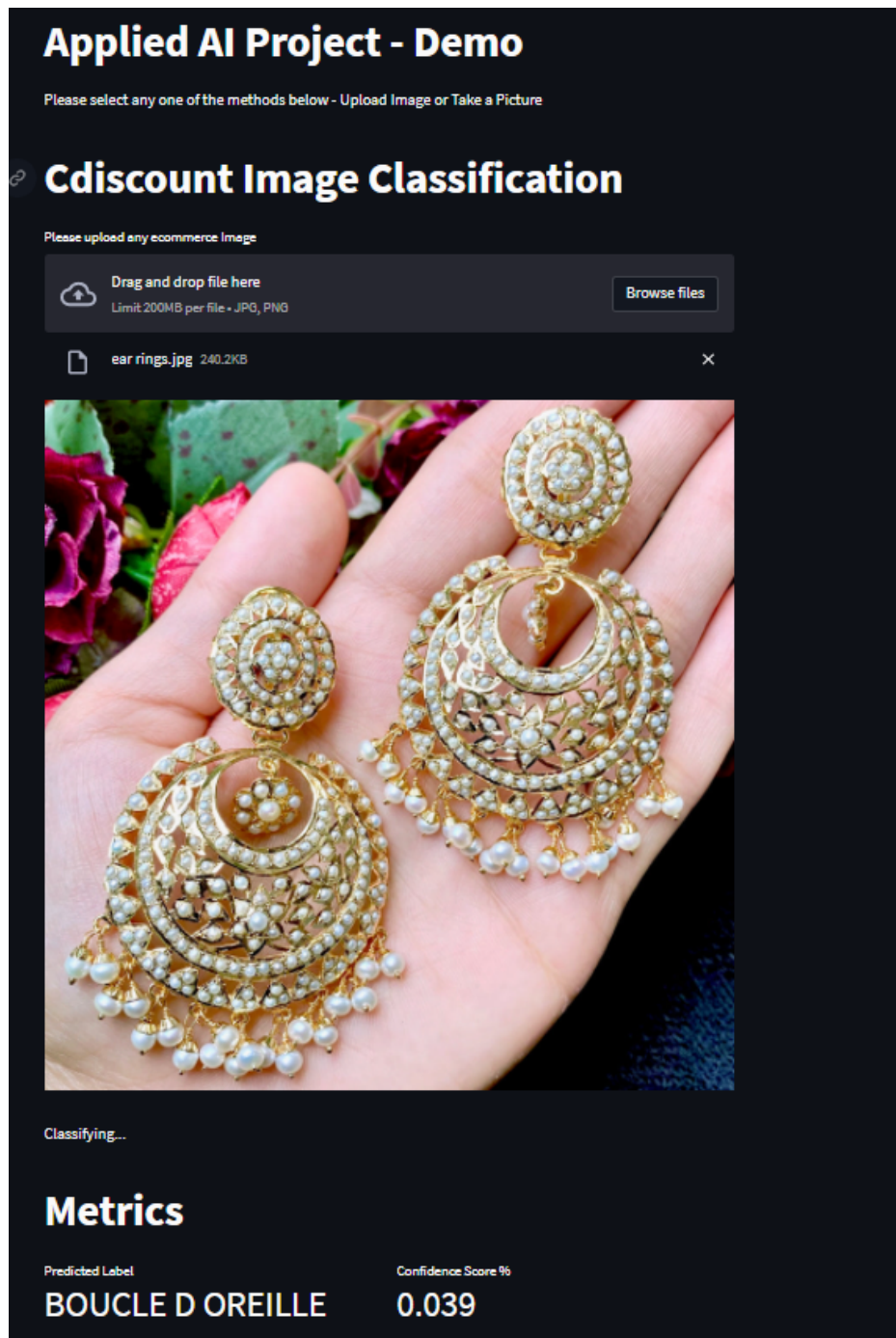
For this we have created two python file:-

1. App.py- which consist of code with respect to creating UI of stremlit application and calling our model for prediction
2. Img_classification.py- this file consist all the preprocessing function and model predict function that is required to give correct output
3. We will run the App.py file using command- *streamlit run app.py*. This will open up a new webpage with you stream lit application.

4. Access the above following below github link:-

- a. App.py- [Cdiscount-Webapp_deployment_heroku/app.py at main · sarthakkaushik/Cdiscount-Webapp_deployment_heroku \(github.com\)](https://github.com/sarthakkaushik/Cdiscount-Webapp_deployment_heroku/blob/main/app.py)
- b. Img_classification.py
- [Cdiscount-Webapp_deployment_heroku/img_classification.py at main · sarthakkaushik/Cdiscount-Webapp_deployment_heroku \(github.com\)](https://github.com/sarthakkaushik/Cdiscount-Webapp_deployment_heroku/blob/main/img_classification.py)

After running the code the Streamlit application would look like below.



3. Deploying Streamlit App on web using Heruko

I would have deployed the model to heruko , but due large size of the model I wasnt able to upload the model to my github hence was not able to link my github repository to heruko server.

But If we have to deploy the model we can easily follow the step mentioned in below URL- [A quick tutorial on how to deploy your Streamlit app to Heroku. | by Navid Mashinchi | Towards Data Science](#)

CONCLUSION

Thus we saw step by step method how we approached the image classification tasks from data collection to preprocessing to using SOTA models employing transfer learning and deploying the model as a webapp using streamlit.

For future courses we could try to improve the model training time using the below method.

Converting bson data to tfrecord so that leverage all the parallel processing available as part of tf.data

Using mixed precision training

Training on multiple GPUs

References

1. https://github.com/GlastonburyC/Cdiscount-Kaggle-Competition/blob/master/bgen_inceptionV3.py
2. <https://github.com/GreensboroAI/CDiscountKeras/blob/master/KerasHumanAnalog.py>
3. <https://github.com/creafz/kaggle-cdiscount>
4. <https://techblog.cdiscount.com/cdiscount-image-dataset/>
5. <http://yutarochan.github.io/assets/pdf/CDiscount%20Kaggle%20Competition.pdf>
<http://neuro.cs.ut.ee/wp-content/uploads/2018/02/cdiscount1.pdf>
6. <https://arxiv.org/pdf/2008.05756.pdf>
7. <https://arxiv.org/pdf/2008.05756.pdf#:~:text=Accuracy%20is%20one%20of%20the,computed%20from%20the%20confusion%20matrix.&text=The%20formula%20of%20the%20Accuracy,confusion%20matrix%20at%20the%20denominator.>

8. <https://stackoverflow.com/questions/60725812/calculating-accuracy-for-multi-classclassification>
9. <https://www.arxiv-vanity.com/papers/2008.05756>