Post Graduate Diploma Thesis

Categorize e-commerce photos: Cdiscount’s Image Classification Challenge

Project Number: 106

Uday Girish Maradana

Post Graduate Diploma in Artificial Intelligence (PGD-AI)
University of Hyderabad in association with Applied Roots

February 27, 2022

Mentored by Applied Roots
Lectures by Srikanth Varma chekuri

Abstract

Product Recommendation and classification is one of the important components for Ecommerce Websites and companies. This greatly helps them in two ways both result in profit and customer likeness. One is helping the customer to find products similar to his liking and the other is creating a database to cluster products based on similarity to suggest more products or options to customers. The problem with the clustering of Ecommerce data or recommendations involves a lot of data handling. There are several sorts of data included in this problem such as Image, Text, Tabular. In this project, we’re gonna explore one such problem to identify the Label of the image to map it to a category level to help in the product classification and categorization.

Here we use one of the big datasets posed as a problem several years back by Cdiscount which is known as Cdiscount’s Image Classification Challenge. Here I have tried multiple Deep learning-based approaches to build a better classification model and also explore multiple optimization methods in terms of optimization of loss.

Contents

Chapter 1

Introduction

What is the problem all about?

CDiscount is France's largest non-food e-commerce company. The company sells everything from TVs to Trampolines and the list of products is rapidly growing. They have seen an increase from 10 million products to 30 million products in 2 years. Ensuring that so many products are well classified is a challenging task.

So Cdiscount applies ML algorithms to text descriptions of products in order to predict their category. As this method seems already close to the maximum potential, Cdiscount believes the next improvement would be classified with respect to Product Images.

This challenge is a part of Kaggle's competition with prize money of 35000 USD for the best solution submission.

Why is this an important problem to solve?

As classifying products either for Recommendation or Clustering or Grouping for further analyzing them to help Business take a decision is highly important, it is required to have a highly accurate Quantitative process to ensure the numbers (Analytics outputs) actually adds value to the Business.

So here this problem is all about attempting to go that extra mile in improving the current quantitative analysis process which is currently on the text-based categorization. The Image level classification is highly relevant as this will definitely help us understand the product and categories much better.

Chapter 2

Literature Survey

Business/Real-world Impact of solving the problem
Business can take better decision or develop advanced system on Recommendations if the categorical classification of the product is better. Impact to real world would be exploring a Heavy E-commerce data Multi class classification problem using Current SOTA Image Classifiers or doing further Research on the same which can add value to overall research community and also aid people who are trying to solve similar problem statements.

After reading some research around the problem statement this directly comes under the Multi Class Image classification. After going through Kaggle kernels, Medium Blogs and Open source implementation some of the approaches which felt can relate to the problem statement are:

1. CNN based approaches - Custom Model development
2. CNN based approaches - Transfer learning + fine tuning
3. CNN based approaches - Using Pretrained Networks Custom Model on top of it
4. Ensemble approaches - Using combined pretrained networks or Stacking Ensembles for Deep learning
5. Optimization and Hyperparameter tuning

The primary aim is to develop an Image classifier based on SOTA models (Older and newer) starting from using ResNet , Inception V3 to very recent Efficient Det . For reference to these individual research please find the references.

One of the core problem in this project is to process large data which is having almost greater than 50 GB of train data and 10 GB of test data. So some of the problems which I found after referring to the previous literature and works are :

1. As the dataset is huge, training directly or writing code to efficiently load the data is a problem but that can be handled with train_example.bson.
2. Training with larger batch sizes might not be possible.
3. Compute Power - Needs a very good GPU

4. Training iterations would be limited so I have to carefully select the approaches.
5. High class imbalance issues have to be dealt with and model analysis or comparison would be difficult because of this.

Some of these problems are handled very well by previous participants in the competitions through one of the four methods below:

1. Writing a Custom Data Generator class using Keras Data Generator class.
2. Using MongoDB based server methods and build a generator on top of it.
3. Converting images to Tensorflow records and train on them.
4. Saving the images into a folder structure and train.

Out of all approaches available around that time people used ResNet for initial predictions. And moreover to get better accuracy they used Multi domain learning such as learning partially from text with the full learnings from Image. This helped them in gaining the extra hand in the project. But for this project we have only considered getting good accuracy with the Image data alone.

Coming to the metrics for optimization used by the previous competition participants and the researchers worked on this dataset before are as below:

Key Metric to Optimize:

a. Business metric definition
   The Business metric is the categorization accuracy of the predictions (Percentage of products which are correctly classified)
b. Why is this metric used ?
   It is quite logical to choose categorization accuracy which is the percentage of products which are correctly classified w.r.t to total number of products.
   In most of the Classification problems, Business uses this as a metric as it gives understanding on both how good the class predictions are and which the class model is not able to predict correctly.

c. Alternative metrics that can be used ? Why are they not preferred in this case?
   Yes there are other alternative metrics that can be used in Multi class classification problems like this.
   Some of the important ones are:
   1. Precision
   2. Recall
   3. F1 Score

4. ROC AUC Score (TPR and FPR)

Especially in this case taking into account the class imbalance F1 Score would be the best metric.

So there are other metrics too to judge the solution approaches especially in multi class classification to compare on classifier with respect to another classifier we can use the below methods:

1. Matthew’s Correlation Coefficient
2. Cohen’s Kappa Score
3. Multi class log loss score (least suggested)

d. Pros and cons of the metric used.

Business Metric (Categorization Accuracy):

Pros:

1. Very easy to calculate
2. Quite logical to understand and straight on point to make a business insight.

Cons:

1. Can’t explain the model or solution completely.
2. Needs help of other metrics such as F1 score to give better insight on Model performance and Comparison
3. This can’t be served as a training metric and this is only a validation metric.
4. This metric is calculated just based on whether the category is correctly predicted or not and doesn’t give insight on the category misprediction analysis.

e. Where does this metric fail ? Where should it not be used ?

1. This category is usually used in most of the classification problems to show the end solution performance to the Business people.
2. This metrics fails to account to give a proper understanding on the model level and category level misprediction comparison
3. This can’t be used in case of problems where class level precision and recall is highly important such as Clinical data Classification, Financial Risk data Classification etc.

f. Where(on what type of problems is this metric used elsewhere in ML and Data Science?)

i. This is mostly common in Multi class classification as a Business metric.

ii. These types of metrics are common in Binary classification too and there it is equivalent to standard accuracy metric.

g. Code: -

```
        Pseudo_Code
   i.   Y_true = df_y_true or dict_y_true
  ii.   Y_pred = df_y_pred or dict_y_pred
 iii.   output_class = []
            for i in Y_true.keys():
                If Y_true[i] == Y_pred[i]:
                    correct_class.append(1)
                else:
                    correct_class.append(0)
  iv.   Categorization_accuracy = sum(output_class)/ len(output_class)
```

What are the requirements the solution must meet or the targets we need to acheive?

1. The solution must be having a good categorization accuracy.
2. To minimise overfitting on some classes because of the imbalances and other issues in the dataset.

Chapter 3

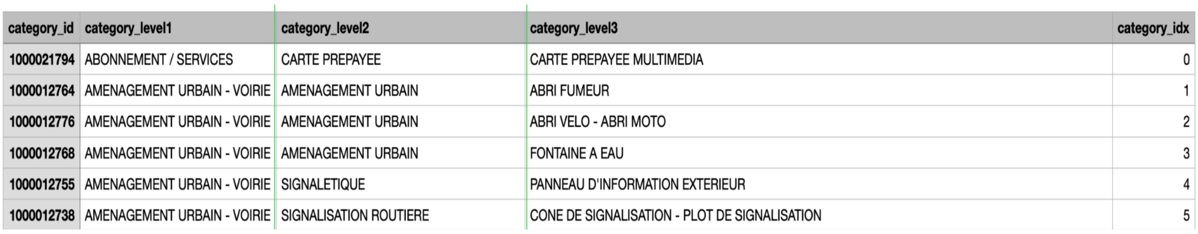
Data Acquisition

This chapter deals with the current data understanding and the sources.

a. Source of the dataset
    - The current source of the data is through the Kaggle CDiscount Competition
    - The dataset is from the Cdiscount Product database.
    - 4 files - train.bson, test.bson, category_names.csv, train_example.bson
    - MongoDB Database dump file

b. Explanation of each feature and datapoint available
    - So with 12 M images of 7M products with 5270 categories this is one of the biggest Image dataset for a Multi class classification problem.
    - The dataset is organized to a 3-level classification tree with categories labeled in French.

Category Level	Cat I	Cat II	Cat III
Nb of Categories	49	483	52700



category_id	category_level1	category_level2	category_level3	category_idx
1000021794	ABONNEMENT / SERVICES	CARTE PREPAYEE	CARTE PREPAYEE MULTIMEDIA	0
1000012764	AMENAGEMENT URBAIN - VOIRIE	AMENAGEMENT URBAIN	ABRI FUMEUR	1
1000012776	AMENAGEMENT URBAIN - VOIRIE	AMENAGEMENT URBAIN	ABRI VELO - ABRI MOTO	2
1000012768	AMENAGEMENT URBAIN - VOIRIE	AMENAGEMENT URBAIN	FONTAINE A EAU	3
1000012755	AMENAGEMENT URBAIN - VOIRIE	SIGNALETIQUE	PANNEAU D'INFORMATION EXTERIEUR	4
1000012738	AMENAGEMENT URBAIN - VOIRIE	SIGNALISATION ROUTIERE	CONE DE SIGNALISATION - PLOT DE SIGNALISATION	5

- And one more thing to notice about the dataset is each product can have a distinct number of images associated with it. You can find the same information in the below table.

Nb of images	1	2	3	4
Nb of Products	4369441	1128588	542792	1029075

- So overall we count precisely 12,371,293 images for 7,069,896 products.

c. Data Size and any challenges you foresee to process it ?

- As far as the data concern the data is huge and heavy class level imbalance is also present.
- The important challenges with the data preprocessing part:
    - Unbalanced Categories
      The categories in the images are highly unbalanced which can create a lot of problems.
      This happened because of the reason that the aim of categorising them is mostly focused towards gathering products with similar characteristics and purposes.

    - Duplicated Images
      As a Reseller can use similar if not identical images to describe several of their products. There is a higher chance that some of the products are with distinct characteristics with identical appearances.

- So processing this unbalanced dataset and making batches for training by creating a custom generator is a task.
- Some of the other issues were noticed with labelling of the dataset as it is semi-automated where clustering is applied to classify some products based on a confidence score. And according to a survey, the overall rate of bad classification is around 10% in each category.
- There are some other problems including multiple class segments presenting in the same images i.e multiple images of the same products with different views or colors present in the same image sample.
- As the data size is huge there is a problem with training a model with a lesser computing power system. Even if Google Colab Pro is used we are limited by the training times.

d. Tools (Pandas, SQL, Spark etc) that you will use to process this data.

- Mostly I will be using Pandas / Spark or with MongoDB to process the data.
- But the higher contribution would be Pandas in processing the data. This is according to the discussions in Kaggle and previous experience. But once after starting to write more optimised code these decisions might change.

e. Data Acquisition:

   i. Open Source data (or) via API (or) via scrapping (or) Generate synthetic Data

      - Open Source Data from Kaggle Competition /Challenge.

   ii. Can you acquire more data in addition to your primary source? If so, how?

      - As the data size is huge maybe we might not need to.
      - But if needed to solve some core data issues like Unbalanced classes I will be using Augmentation.
      - But if needed we can handpick the data but this is a time taking process which most probably not suggested.

Chapter 4

EDA & Feature Extraction

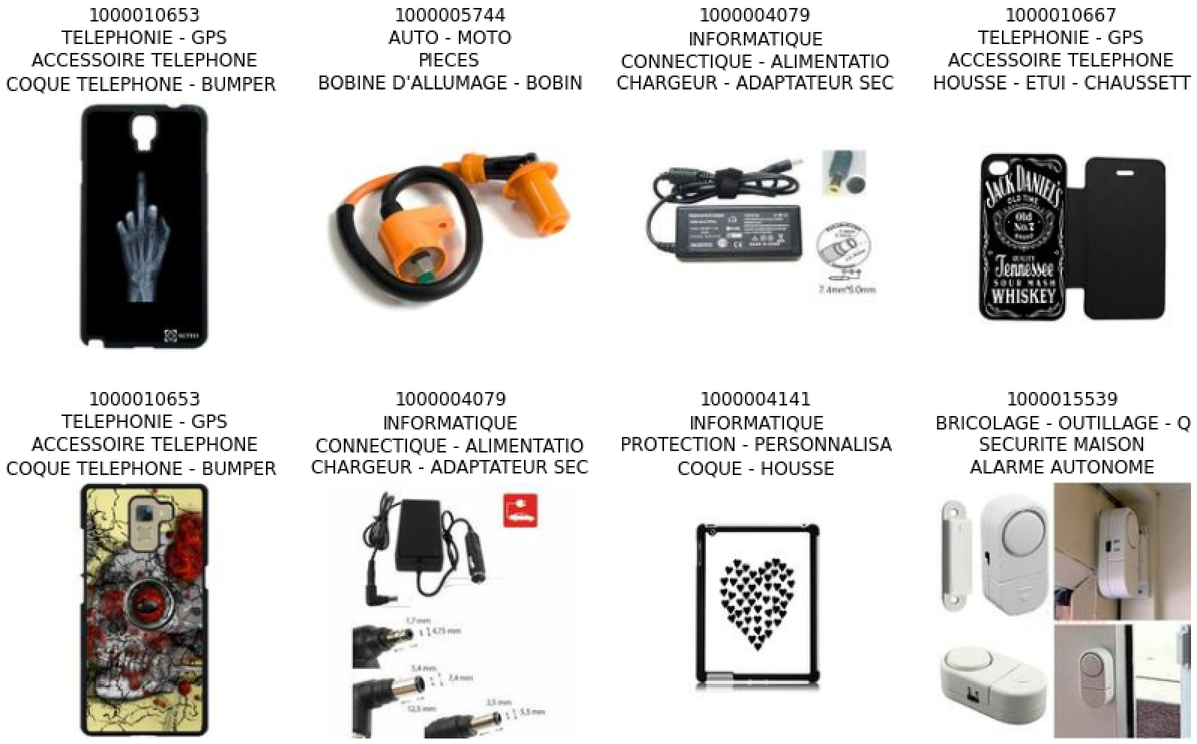
Exploratory Data Analysis :

Initial Observations:

1. As the problem statement is related to the image dataset there is very less to do on the EDA Part.
2. Image Datasets usually don't have any Univariate/ Multivariate analysis as the features are the pixels themselves and usually highly non linear.

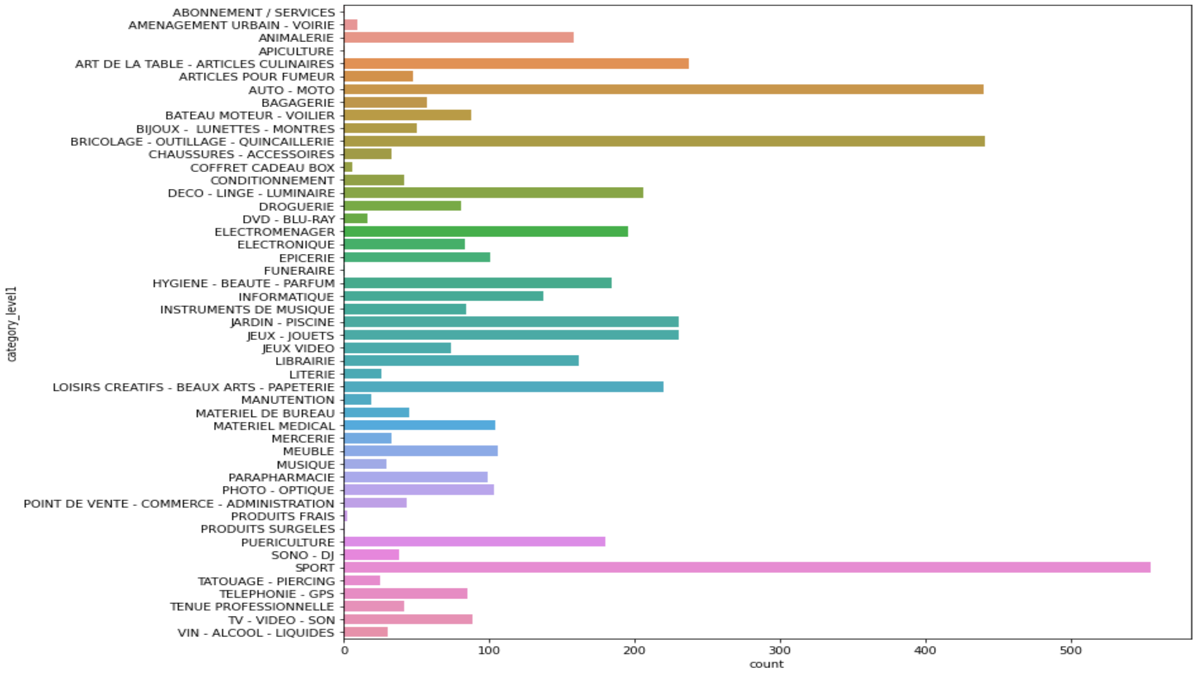
Some Image Samples and Data Distribution Visualisations:

1. Some sample images with classes and product ids from the train set.



2. Some sample images with classes and product ids from the test set.

Category_1 Distribution Visualisation

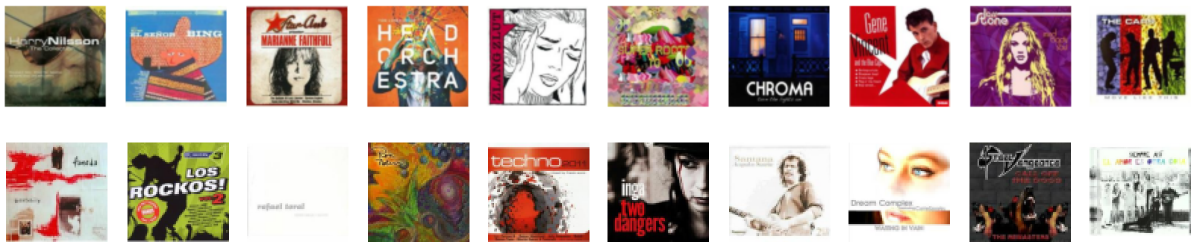


Category_2 and Category_3 have higher category entries to be plotted and visualised properly. So to just point out here Category_2 and Category_3 have 483 and 5263 entries respectively.

Most frequent category in the train set is
[[1000018296 'MUSIQUE' 'CD' 'CD POP ROCK - CD ROCK INDE']]

Visualisation of most frequent category:

1000018296
MUSIQUE
CD
CD POP ROCK - CD ROCK INDE

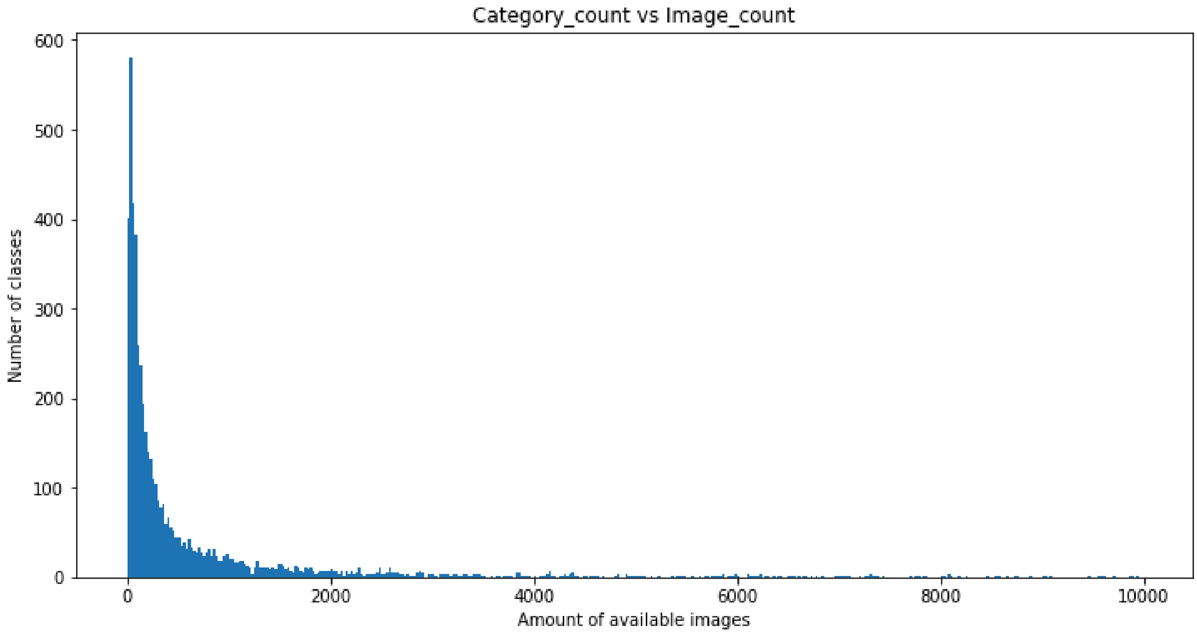


Visualisation of some less frequent categories:

1000000522
PUERICULTURE
SOMMEIL BEBE
FLECHE DE LIT BEBE

1000000896
EPICERIE
CONSERVE DE LEGUME
POIVRON EN CONSERVE

Distribution plot of Number of categories that are presented by an amount of images:



Problem 1 ⇔ Class Imbalance:

According to previous literature review and also Notebook level data loading and analysis there is high class imbalance in the data.

And this graph helps us understand the current data imbalance in the dataset.

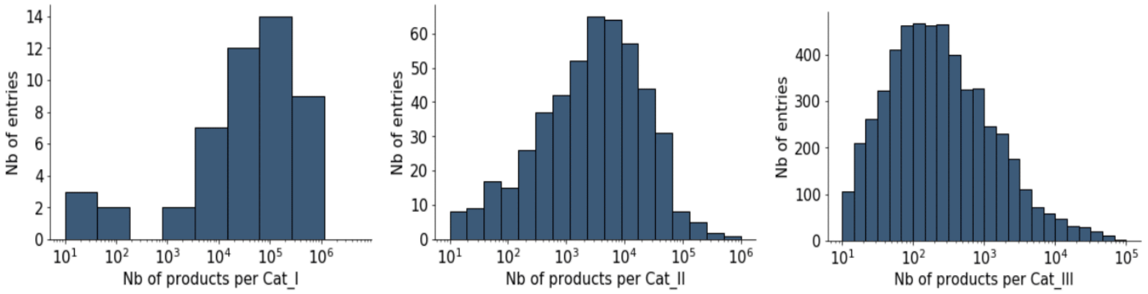


Figure 2: Distributions of products per category for each category level. From left to right: Cat I, Cat II and Cat III.

We have already studied that there are three primary categories in the dataset. And from the above graph we can see the distribution of products per category.

Due to this class imbalance there is a definitive need to balance the Train and Validation datasets.

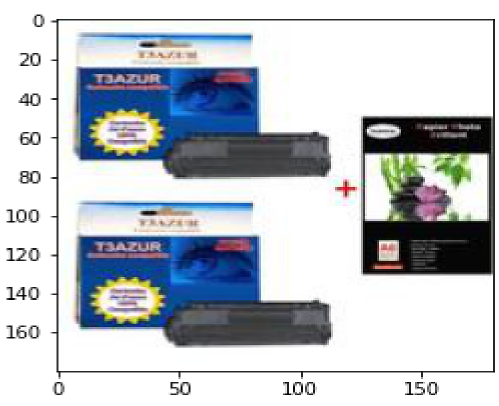
Problem 2 ⇔ Images with Duplication, No Content, SceneInImage, etc.

There are instances where multiple segments of the same products are identified in the same image.And also there are issues such as null images with No content in it and also with multiple products in single image, multiple views of the same image etc.

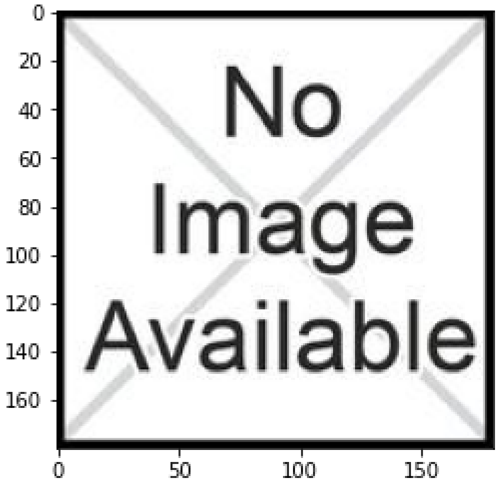
This analysis is done on viewing different images manually on iterating over two batches of 128 images each selected at random from the training set.

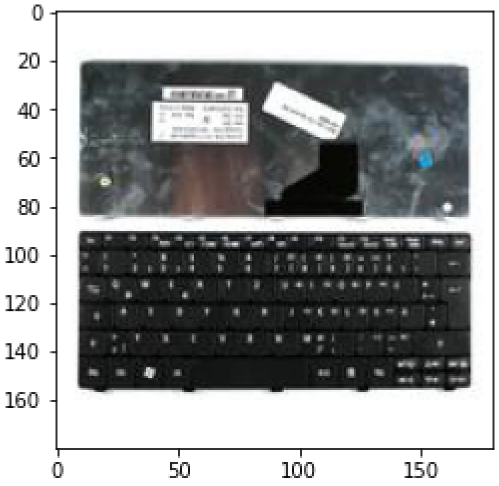
You can view the same list below:

1. Images with Multiple instances of Same products:

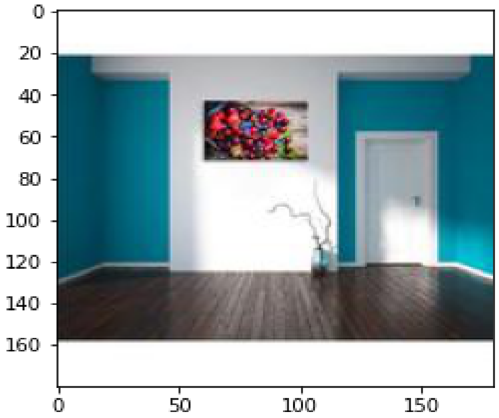


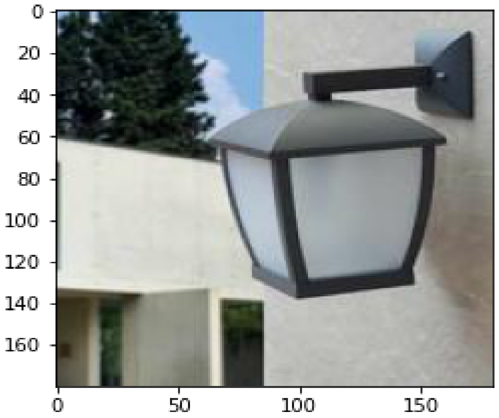
2. Images with No content or Noise



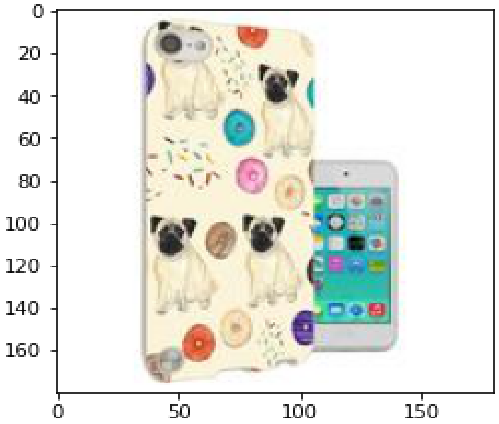


3. Images with Product in Complex Scenes (Multicolor Scene backgrounds)

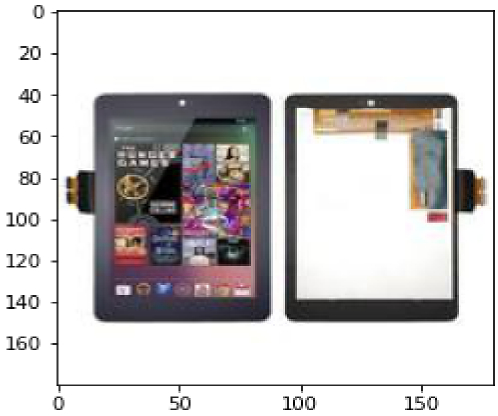




4. Images with Multiple products in a single product list



5. Images with Parts of the Products or Multi view



The above analysis helps us to understand how complex the dataset and those above can have a significant impact on the variance of the model.
Creating a model on this sort of dataset is pretty challenging to ensure both high accuracy and equally generalizing on all the products.

Problem 3 ⇔ Data Loading - Creating a BSON Generator

After going through several implementations and Discussions in Kaggle it is found to be using a BSON Generator class to make a iter method would be better to pass this method while training.

There are two issues identified in the implementation of the famous discussion.

1. Use of Keras Preprocessing Image load function
    a. There are some issue to read direct bytes or use IO buffer to read the data as the implementation is a bit broken
2. Using PIL Image instead of Keras Preprocessing
    a. Using PIL Image Open function solved the issue but created a little CPU unused time problem because of the file opening and reading

Explored Concepts for Data loading:

1. Implementation of Multiprocessing in Generator - Exploration
2. Checking the Keras preprocessing dependencies (Version) to use the load_img function.
3. Alternative approach exploration with Dataloader in PyTorch.

Note:

To go through the implementations or the Code used please follow the below links: Please don’t edit them just for representational purposes and most of the code base is understood and taken from Kaggle Code and Discussions of the same competition.

Please check the notebooks associated with this document to get to know more about the work till now.
The notebook names are:

1. CDiscount_Analysis_Base_Kaggle_Copied_kernel_UG_Phase2.ipynb
    a. About basic creation of Image Generator class for next modelling purposes.
2. Applied_AI_Proj_Notebook_Data_Visualisation_Phase2.ipynb
    a. About Data Analysis on both train and test
    b. Some Train and Test data visualisations

Chapter 5

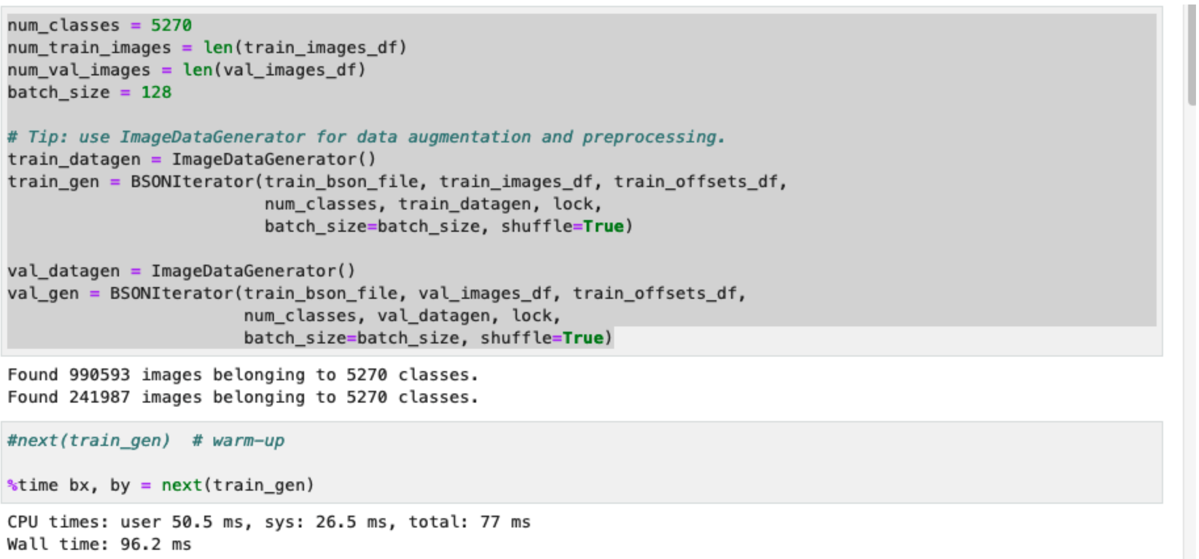
Modelling & Error Correction

Phase: Intermediate Explorations

Here two methods are especially tried one of which is the best data loading method in the discussions page of Kaggle's challenge which is the Data Augmentor class which is augmented with the help of a BSON iterator class.

Data loading is a bit slow and further methods are being explored on how to boost the speeds.
Other ways of Data loading using a MongoDB server are explored on my local system which is comparably at least two times faster than the current method but because of some other issues (bug) in the code, it is not used at current.

Now the approach used is a Data Augmentor Class.
The below-shown figure gives us an understanding of the current loading speed for a batch of 128 with an image size of (180,180)
Including two data generator transforms such as Random Transform and Image Standardization.



```
num_classes = 5270
num_train_images = len(train_images_df)
num_val_images = len(val_images_df)
batch_size = 128

# Tip: use ImageDataGenerator for data augmentation and preprocessing.
train_datagen = ImageDataGenerator()
train_gen = BSONIterator(train_bson_file, train_images_df, train_offsets_df,
                         num_classes, train_datagen, lock,
                         batch_size=batch_size, shuffle=True)

val_datagen = ImageDataGenerator()
val_gen = BSONIterator(train_bson_file, val_images_df, train_offsets_df,
                       num_classes, val_datagen, lock,
                       batch_size=batch_size, shuffle=True)
```

```
Found 990593 images belonging to 5270 classes.
Found 241987 images belonging to 5270 classes.
```

```
#next(train_gen)  # warm-up

%time bx, by = next(train_gen)
```

```
CPU times: user 50.5 ms, sys: 26.5 ms, total: 77 ms
Wall time: 96.2 ms
```

So the loading time for a batch of 128 is 96.2 ms in wall time.

Modelling:

Some of the currently explored approaches including Xception, Inception - V3, Efficient Net-B7.

Out of all Efficient Net- B7 performance is good till now.

Frameworks Used: Tensorflow, tf.keras, Keras

Detailed Discussion:

Exception:

Xception is a Depth Wise Separable Convolutions-based architecture and having nearly the same parameters as Inception V3.
Convolutions and depthwise separable convolutions lie at both extremes of a discrete spectrum with Inception modules being an intermediate point in between. This observation is the main reason to come up with an architecture replacing Inception modules with depthwise separable convolutions.

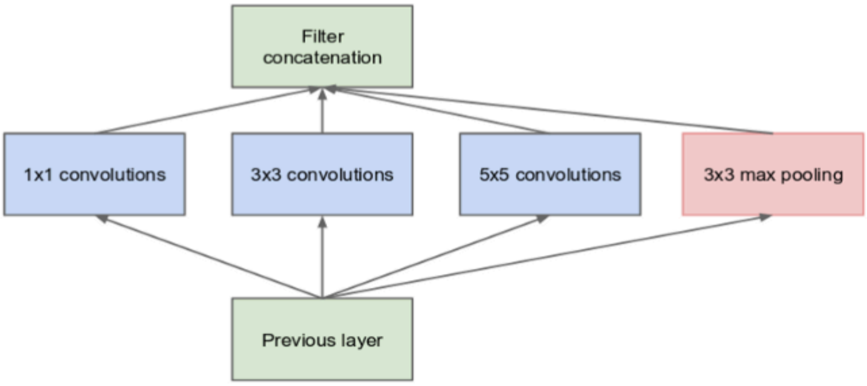
This is the first exploration I did use a Keras example to train it from scratch which is a little bit altered architecture.

After training for some 10 epochs it is observed that the train and Val accuracy is around 40% and 42%.

After this, I have started to try the methods suggested in the Kaggle discussions and SOTA.

Inception V3:

Inception architectures are mostly based on Inception modules especially having filters with multiple sizes operating on the same level. This helps the model in exploring the data wider.
When we design Deep networks adding these modules helps in adding the wider exploration aspect.

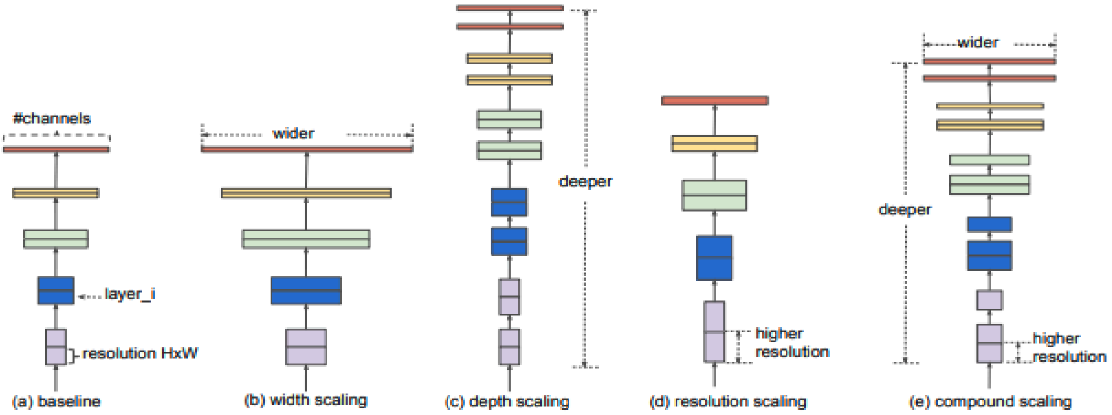


(a) Inception module, naïve version

Even though running the InspectionV3 with fine-tuning gave Train and Val accuracy around 53 and 45 percent by the end of 10 epochs. Currently working on Hyperparameter tuning and tinkering how many layers to train.

Efficient Net B-7

The efficient implementations are based on the paper "*Efficient Net: Rethinking Model Scaling for Convolutional Neural Networks".*
This paper explains more on the study on ConvNet Scaling and identifying the architecture than can carefully balance network width,depth and resolutions. This paper proposes to use a Compound Scaling method which enables us to easily scale ConvNets.

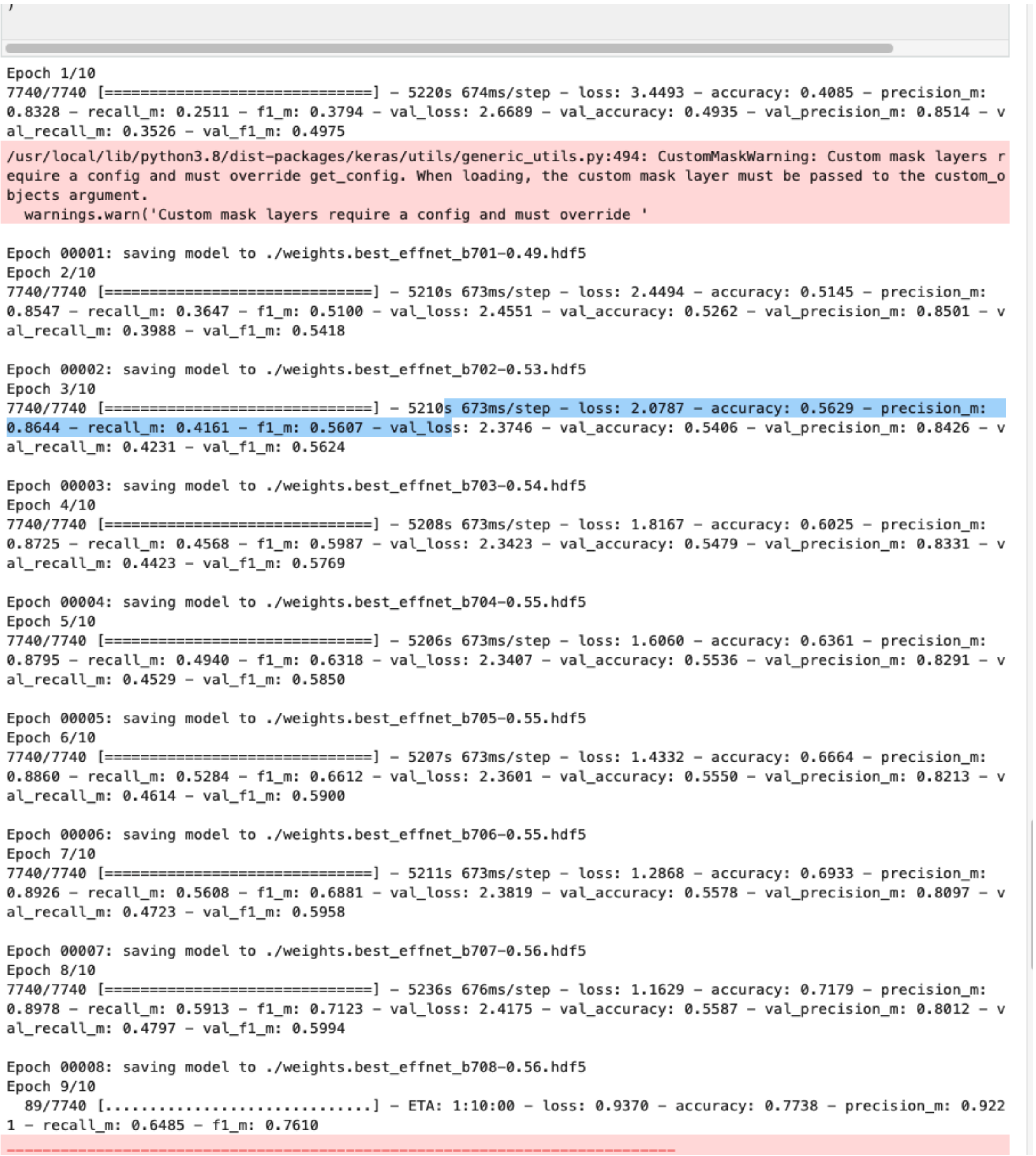


(a) baseline (b) width scaling (c) depth scaling (d) resolution scaling (e) compound scaling

The picture shows the different scaling which we can do a baseline convnet to scale easily to fit and generalize better. The compound scaling which is the combination of most of the other scalings in a single network is what is used by Efficient Nets.

This is the baseline model on which I'm trying to improve.

Previous training for 8 epochs resulted in train accuracy of 71.79 % and val accuracy of 55.78 percent.



```
,
Epoch 1/10
7740/7740 [==============================] - 5220s 674ms/step - loss: 3.4493 - accuracy: 0.4085 - precision_m:
0.8328 - recall_m: 0.2511 - f1_m: 0.3794 - val_loss: 2.6689 - val_accuracy: 0.4935 - val_precision_m: 0.8514 - v
al_recall_m: 0.3526 - val_f1_m: 0.4975
/usr/local/lib/python3.8/dist-packages/keras/utils/generic_utils.py:494: CustomMaskWarning: Custom mask layers r
equire a config and must override get_config. When loading, the custom mask layer must be passed to the custom_o
bjects argument.
  warnings.warn('Custom mask layers require a config and must override '

Epoch 00001: saving model to ./weights.best_effnet_b701-0.49.hdf5
Epoch 2/10
7740/7740 [==============================] - 5210s 673ms/step - loss: 2.4494 - accuracy: 0.5145 - precision_m:
0.8547 - recall_m: 0.3647 - f1_m: 0.5100 - val_loss: 2.4551 - val_accuracy: 0.5262 - val_precision_m: 0.8501 - v
al_recall_m: 0.3988 - val_f1_m: 0.5418

Epoch 00002: saving model to ./weights.best_effnet_b702-0.53.hdf5
Epoch 3/10
7740/7740 [==============================] - 5210s 673ms/step - loss: 2.0787 - accuracy: 0.5629 - precision_m:
0.8644 - recall_m: 0.4161 - f1_m: 0.5607 - val_loss: 2.3746 - val_accuracy: 0.5406 - val_precision_m: 0.8426 - v
al_recall_m: 0.4231 - val_f1_m: 0.5624

Epoch 00003: saving model to ./weights.best_effnet_b703-0.54.hdf5
Epoch 4/10
7740/7740 [==============================] - 5208s 673ms/step - loss: 1.8167 - accuracy: 0.6025 - precision_m:
0.8725 - recall_m: 0.4568 - f1_m: 0.5987 - val_loss: 2.3423 - val_accuracy: 0.5479 - val_precision_m: 0.8331 - v
al_recall_m: 0.4423 - val_f1_m: 0.5769

Epoch 00004: saving model to ./weights.best_effnet_b704-0.55.hdf5
Epoch 5/10
7740/7740 [==============================] - 5206s 673ms/step - loss: 1.6060 - accuracy: 0.6361 - precision_m:
0.8795 - recall_m: 0.4940 - f1_m: 0.6318 - val_loss: 2.3407 - val_accuracy: 0.5536 - val_precision_m: 0.8291 - v
al_recall_m: 0.4529 - val_f1_m: 0.5850

Epoch 00005: saving model to ./weights.best_effnet_b705-0.55.hdf5
Epoch 6/10
7740/7740 [==============================] - 5207s 673ms/step - loss: 1.4332 - accuracy: 0.6664 - precision_m:
0.8860 - recall_m: 0.5284 - f1_m: 0.6612 - val_loss: 2.3601 - val_accuracy: 0.5550 - val_precision_m: 0.8213 - v
al_recall_m: 0.4614 - val_f1_m: 0.5900

Epoch 00006: saving model to ./weights.best_effnet_b706-0.55.hdf5
Epoch 7/10
7740/7740 [==============================] - 5211s 673ms/step - loss: 1.2868 - accuracy: 0.6933 - precision_m:
0.8926 - recall_m: 0.5608 - f1_m: 0.6881 - val_loss: 2.3819 - val_accuracy: 0.5578 - val_precision_m: 0.8097 - v
al_recall_m: 0.4723 - val_f1_m: 0.5958

Epoch 00007: saving model to ./weights.best_effnet_b707-0.56.hdf5
Epoch 8/10
7740/7740 [==============================] - 5236s 676ms/step - loss: 1.1629 - accuracy: 0.7179 - precision_m:
0.8978 - recall_m: 0.5913 - f1_m: 0.7123 - val_loss: 2.4175 - val_accuracy: 0.5587 - val_precision_m: 0.8012 - v
al_recall_m: 0.4797 - val_f1_m: 0.5994

Epoch 00008: saving model to ./weights.best_effnet_b708-0.56.hdf5
Epoch 9/10
  89/7740 [..............................] - ETA: 1:10:00 - loss: 0.9370 - accuracy: 0.7738 - precision_m: 0.922
1 - recall_m: 0.6485 - f1_m: 0.7610
---------------------------------------------------------------------------
```

The best epoch is the 3rd and 6th epoch as after which the model began overfitting on train.

On 3rd epoch the train and val accuracy are 56.29 and 54.06 which is very close and best fit post that the model started a bit overfitting.

Currently running hyperparameter tuning and modifying and also experimenting with ensembles to get better results.
Also NASNet Large is currently running post which results will be appended to the next document.

Out of all the models, Efficient B-7 is more generalized and has better architecture as it is recently released and is also one of the SOTA in most computer vision tasks.

Chapter 6

Advanced Modelling & Feature Correction

Explorations Conducted and Work done:

After the initial explorations were done on the modeling part three models were trained extensively such was InceptionV3, Resnet 101, and Efficient Net -B7.

Some of the approaches tried are:

1. Augmentation - Flip
2. Learning rate Decay
    a. Exponential Decay
    b. Polynomial Decay
    c. Cyclical Learning Rate
    d. Cyclical Exponential Learning rate decay etc.
3. Tried with Subsampling of data.
4. Working with Global Max Pooling at the last layer.
5. 1*1 Convolutions at the end of the pretrained network to learn features better.
6. 1*1 convolutions at the end with Batch Norm and dropout to reduce overfitting.
7. Reduce learning rate callback.
8. Different Image shapes - (180,180,3) , (224,224,3)

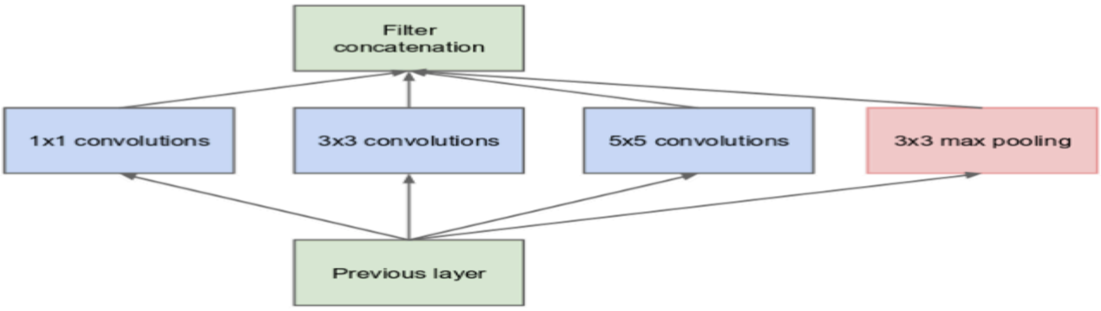
Exception for a sample implementation is implemented from scratch using tf.keras.layers. Other models are directly imported from tf.keras pretrained models. 1*1 convolutions are used to keep the learning information constant according to a filter based learning and also to reduce the number of Trainable params.

Please find some of the results below with respect to each architecture's best performance.

Inception V3:

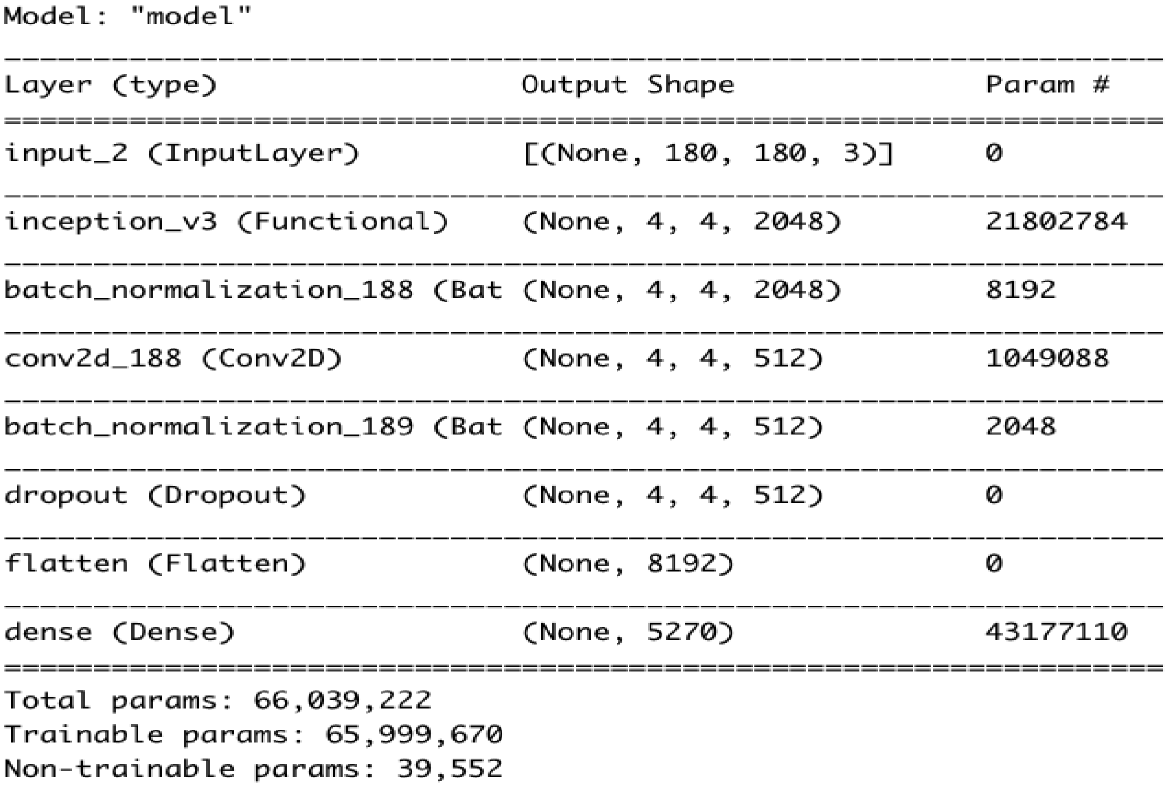
Inception architectures are mostly based on Inception modules especially having filters with multiple sizes operating on the same level. This helps the model in exploring the data wider.

When we design Deep networks adding these modules helps in adding the wider exploration aspect.

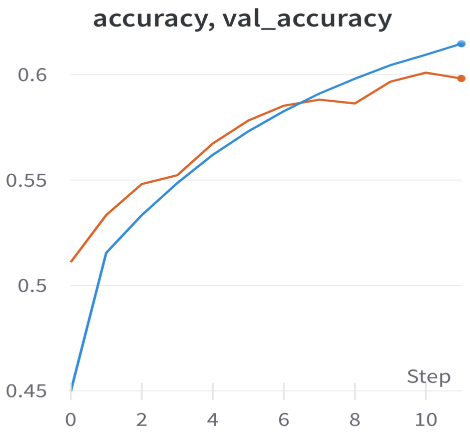


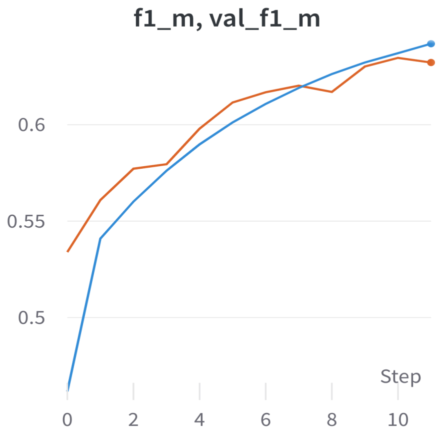
(a) Inception module, naïve version

Inception V3 Training with 1*1 Conv layers Results



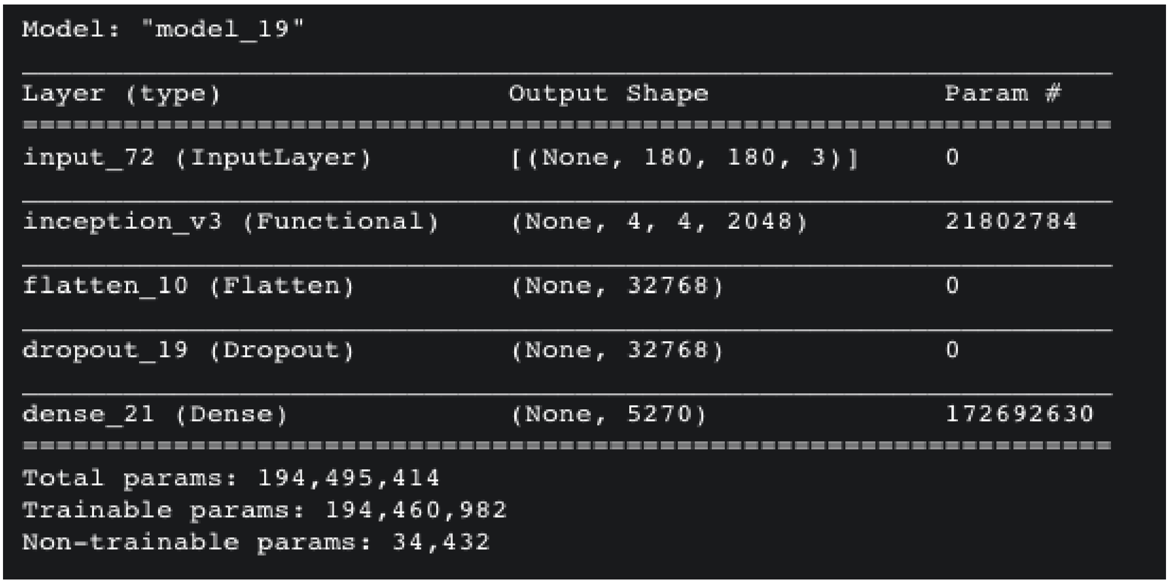
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 180, 180, 3)]     0
_________________________________________________________________
inception_v3 (Functional)    (None, 4, 4, 2048)        21802784
_________________________________________________________________
batch_normalization_188 (Bat (None, 4, 4, 2048)        8192
_________________________________________________________________
conv2d_188 (Conv2D)          (None, 4, 4, 512)         1049088
_________________________________________________________________
batch_normalization_189 (Bat (None, 4, 4, 512)         2048
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 5270)              43177110
=================================================================
Total params: 66,039,222
Trainable params: 65,999,670
Non-trainable params: 39,552
```



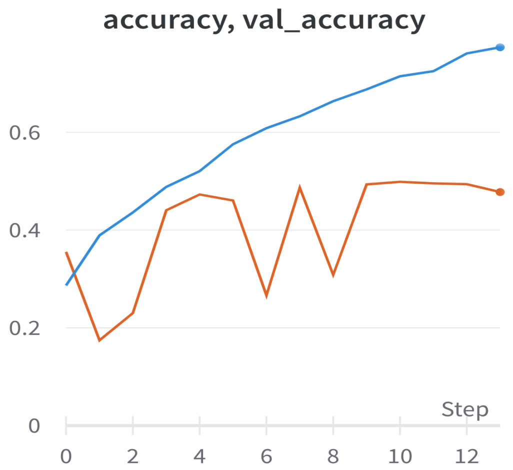


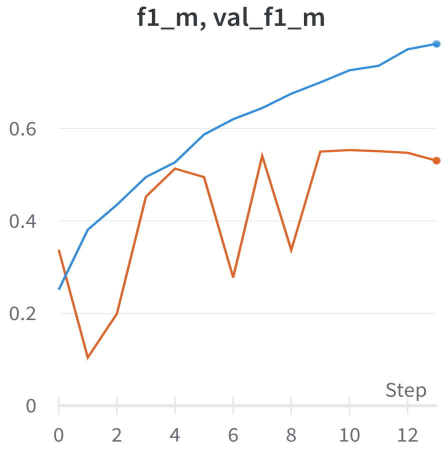
The highest train and validation accuracy we have achieved till now is 60.95 and 60.1 respectively. The highest train and validation F1 with this architecture are 63.71 and 63.47 respectively.

Inception V3 with Flatten



```
Model: "model_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_72 (InputLayer)        [(None, 180, 180, 3)]     0
_________________________________________________________________
inception_v3 (Functional)    (None, 4, 4, 2048)        21802784
_________________________________________________________________
flatten_10 (Flatten)         (None, 32768)             0
_________________________________________________________________
dropout_19 (Dropout)         (None, 32768)             0
_________________________________________________________________
dense_21 (Dense)             (None, 5270)              172692630
=================================================================
Total params: 194,495,414
Trainable params: 194,460,982
Non-trainable params: 34,432
```

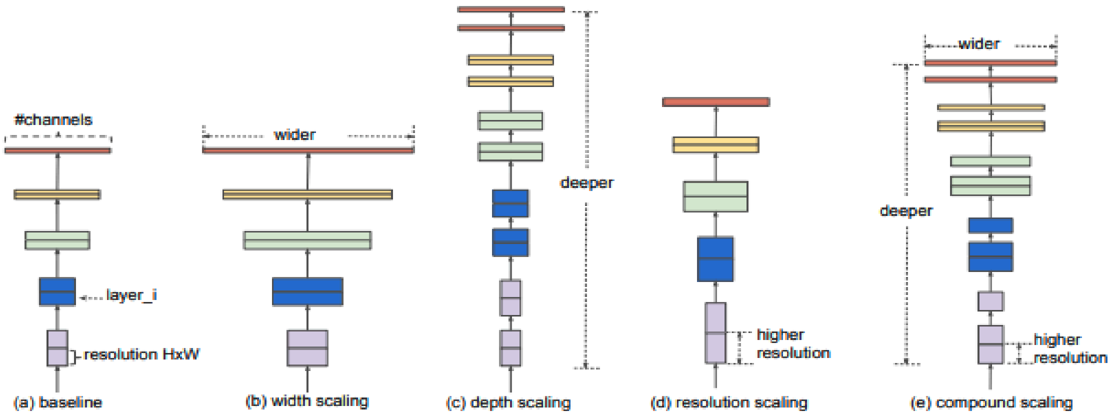




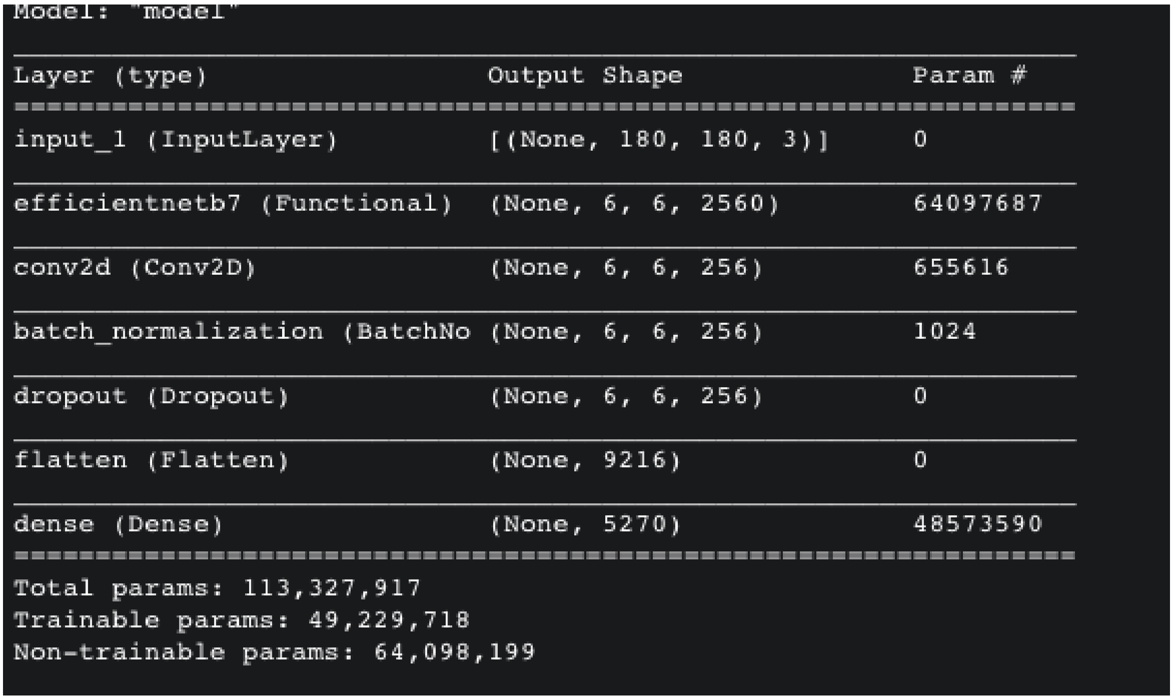
Flattening directly after pretrained networks reduced the accuracy a lot and there is lot of overfitting this was the initial experiments and this can clearly help us to understanding direct Flattening after Pretrained networks won’t work in most of the cases.

Efficient Net B-7

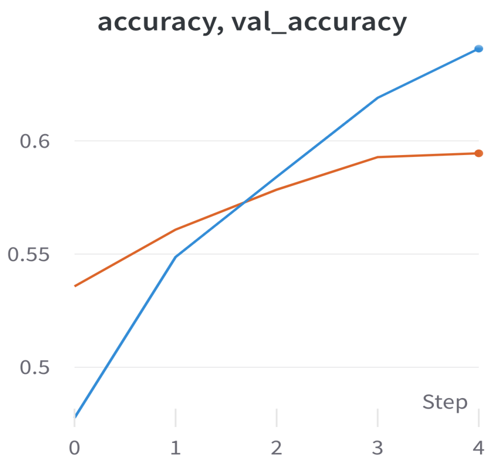
The efficient implementations are based on the paper “*Efficient Net: Rethinking Model Scaling for Convolutional Neural Networks”.*
This paper explains more on the study on ConvNet Scaling and identifying the architecture than can carefully balance network width,depth and resolutions. This paper proposes to use a Compound Scaling method which enables us to easily scale ConvNets.

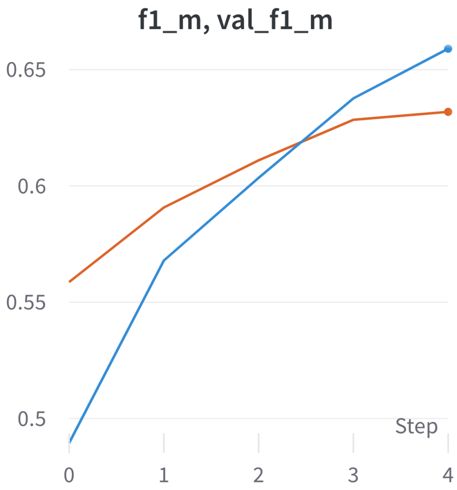


Efficient Net B-7 with 1*1 Conv Results



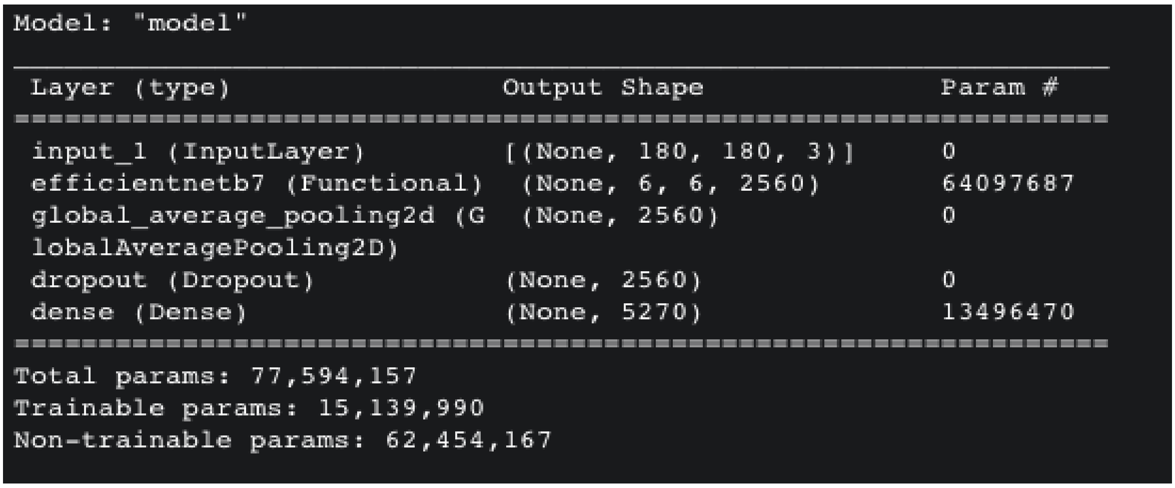
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 180, 180, 3)]     0
_________________________________________________________________
efficientnetb7 (Functional)  (None, 6, 6, 2560)        64097687
_________________________________________________________________
conv2d (Conv2D)              (None, 6, 6, 256)         655616
_________________________________________________________________
batch_normalization (BatchNo (None, 6, 6, 256)         1024
_________________________________________________________________
dropout (Dropout)            (None, 6, 6, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 5270)              48573590
=================================================================
Total params: 113,327,917
Trainable params: 49,229,718
Non-trainable params: 64,098,199
```



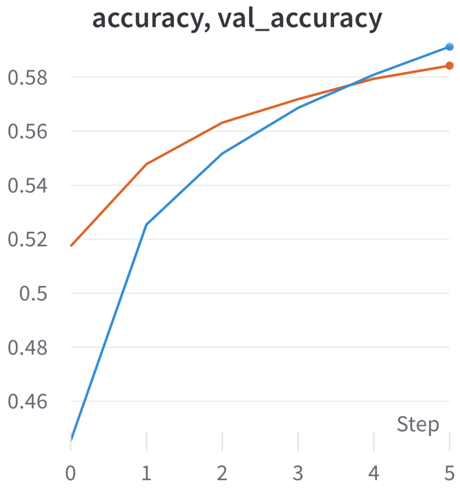


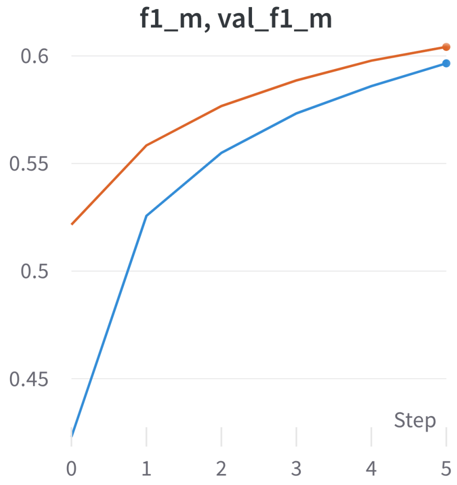
The top validation and train accuracy are 59.28 and 61.9 respectively. The top validation and train F1 are 62.85 and 63.7.

Efficient Net B-7 with Global Max Pooling



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0
 efficientnetb7 (Functional)  (None, 6, 6, 2560)       64097687
 global_average_pooling2d (G  (None, 2560)             0
 lobalAveragePooling2D)
 dropout (Dropout)           (None, 2560)              0
 dense (Dense)               (None, 5270)              13496470
=================================================================
Total params: 77,594,157
Trainable params: 15,139,990
Non-trainable params: 62,454,167
```

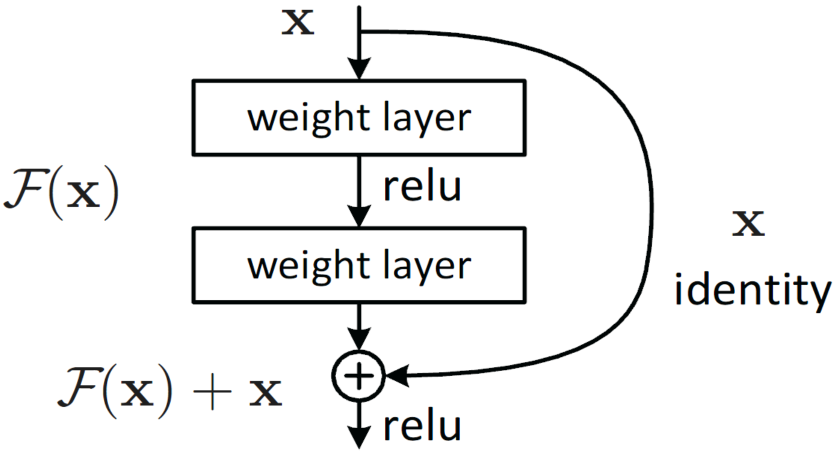




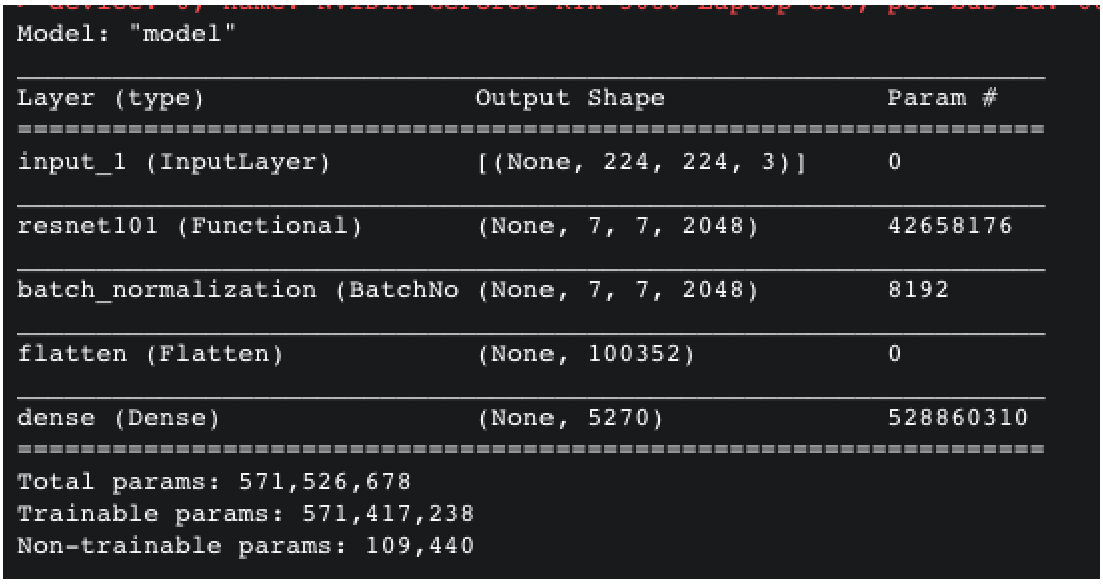
With Global Max Pooling and using transfer learning, we are able to achieve train and validation accuracy of 59.13 and 58.43 respectively. Correspondingly the train and validation F1 are 59.66 and 60.42 respectively.

ResNets - ResNet 101

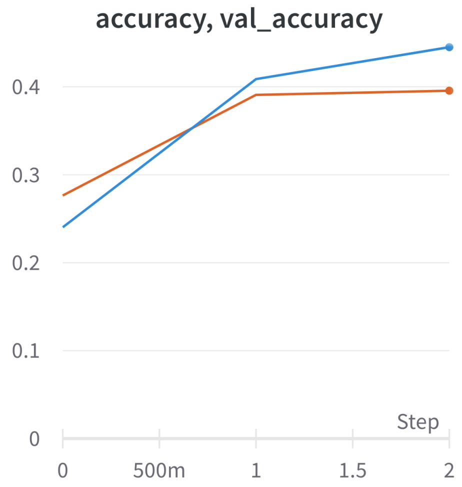
ResNet is a short name for a residual network. When we build Deeper neural networks , when the network starts to converge, a degradation problem in the learning with respect to the network depth where accuracy gets stagnated over a point of time is found. To overcome these sorts of issues and build deeper networks without facing this problem Microsoft introduced residual learning where the previous networks learnings are shared (or) added to the deeper layers to preserve information and reduce the forgetting or over learning.

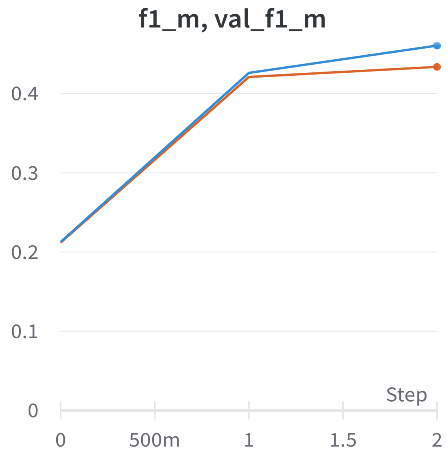


ResNet 101 with Flatten Results



```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
resnet101 (Functional)       (None, 7, 7, 2048)        42658176
_________________________________________________________________
batch_normalization (BatchNo (None, 7, 7, 2048)        8192
_________________________________________________________________
flatten (Flatten)            (None, 100352)            0
_________________________________________________________________
dense (Dense)                (None, 5270)              528860310
=================================================================
Total params: 571,526,678
Trainable params: 571,417,238
Non-trainable params: 109,440
```





Here in the case of ResNet 101, the training is run for only 2-3 epochs on Google collab as the training time is really high on the dataset which is around 8-9 hrs per epoch on the entire dataset and it was observed with direct flattening it is not working so stopped this iteration.

Chapter 7

Deployment

Deployment is one of the crucial component of the entire Machine learning lifecycle. Usually Deployment is more related to the engineering aspect of the Machine learning project.

Deployment involves optimization of the trained model and getting it ready to work in the real environment. There are several parameters on we select the best model to deploy.

Two such metrics are categorised into Train metrics and Inference metrics.

Train Metrics:

Model Performance metrics such as:

1. Train and Val Accuracy
2. Train and Val Precision, Recall, F1
3. AUC and ROC curves etc.

Inference or Model Deployability metrics:

1. Model Size
2. Model State Dict Data type vs Accuracy (Optimization losses in Model accuracy)
3. Model Inference time
4. Model Loading time

Model inference time is again compared with Flops and Memory consumption on GPU or CPU.

Here we selected the best model on F1 Score.

It is a Inception V3 + A custom classification head (Includes a 1*1 CNN and Dense layers) model.

Here we haven't used much of the Optimization the model is a straight loading from the system using Model load from Keras.
The other deployment strategies we can follow are:

1. Tensorflow Serving (CPU based deployment)
2. ONNX Serving (CPU based deployment)

3. TensorRT Server (GPU based deployment)

Here I haven't optimized the model and the inference time is not that problematic so kept the model as it is thereby ensuring the Model performance both in Accuracy and Speed.

An APi is developed on the Model load and predict method using FastAPI and Uvicorn.

And the frontend for the Image Upload and inference submission was developed using Streamlit. The model is always loaded onto the RAM when the FastAPI gets initiated and stays in the RAM to serve the incoming requests from streamlit web app.

The F1 Score of the current model is : 63.4

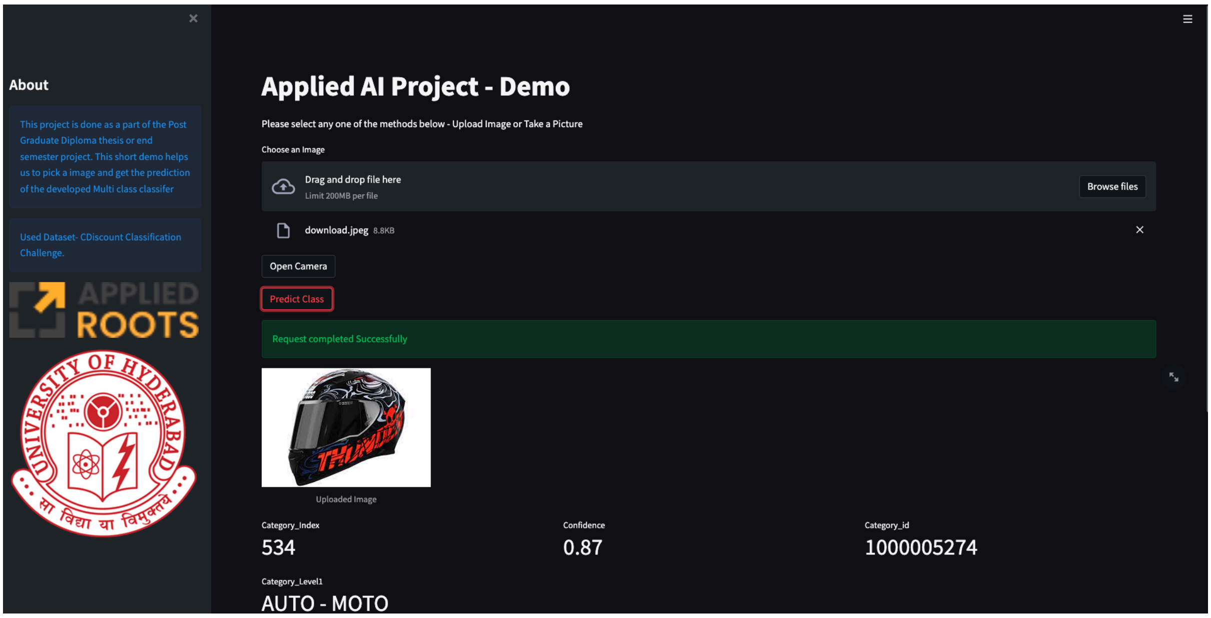
The current Deployment metrics are as follows:

1. API Response Time : 0.5-0.1 s (Depends on Image size and Network latency)
2. Model Inference time: 0.036s

Here the API response time involves three primary components:

1. Image Upload and Sending it to the server
2. Image Resizing to pass it to the model (Can use Dynamic Input using better optimization method - but this is out of scope in the present work, planned to personally explore in the future)
3. Model Inference time.

This is how the Web app looks:



About

This project is done as a part of the Post Graduate Diploma thesis or end semester project. This short demo helps us to pick a image and get the prediction of the developed Multi class classifer

Used Dataset- CDiscount Classification Challenge.

Applied AI Project - Demo

Please select any one of the methods below - Upload Image or Take a Picture

Choose an Image

Drag and drop file here

Limit 200MB per file

download.jpeg 8.8KB

Request completed Successfully

Uploaded Image

Category_Index
534

Confidence
0.87

Category_id
1000005274

Category_Level1
AUTO - MOTO

Input:

Image upload using Browse files or Camera Option.
Currently Browse files is preferred as on Camera option it is still having bugs on streamlit front. Otherwise I'm having plans on implementing WebRTC based stream to also show Video stream based condition but was not able to implement because of the current timeline.

Once you hit the predict class it internally request the FastAPI with a POST request. For testing purposes we can also

Output:

1. Category_Index
2. Confidence
3. Category_Id
4. Category_Level1
5. Category_Level2
6. Category_Level3
7. Response Time
8. Model Inference time

Entire Code is pushed to Git which contains the Train, Model API and Web App. Also DockerFile Train and Docker File for Deployment is given to run the entire pipeline at ease.Because of the File size restrictions the model is not present in Git please try to download it from the below given link.

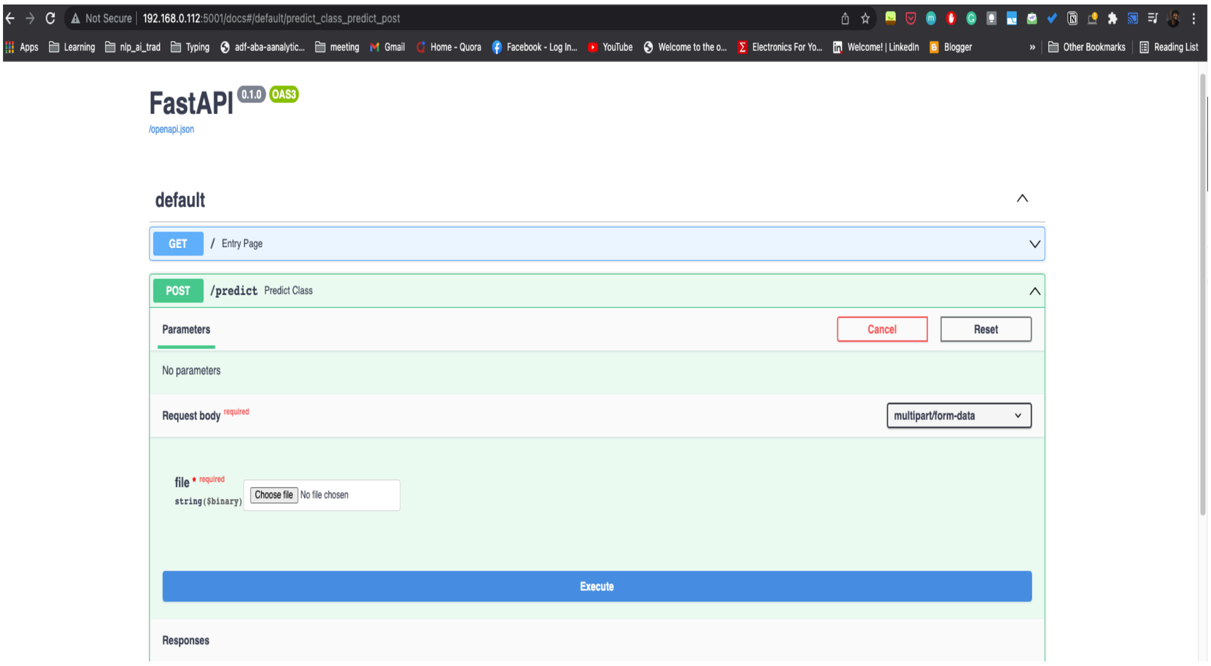
Github Link: https://github.com/udaygirish/cdiscount_challenge_code.git
Model Link:
https://drive.google.com/drive/folders/1i98zK2NdQhVRMKjF1eTxcODQgKLqwrVx?usp=sharing

For downloading the dataset please use kaggle. A script is provided in the repository to download if the Kaggle CLI is already authenticated in your system.

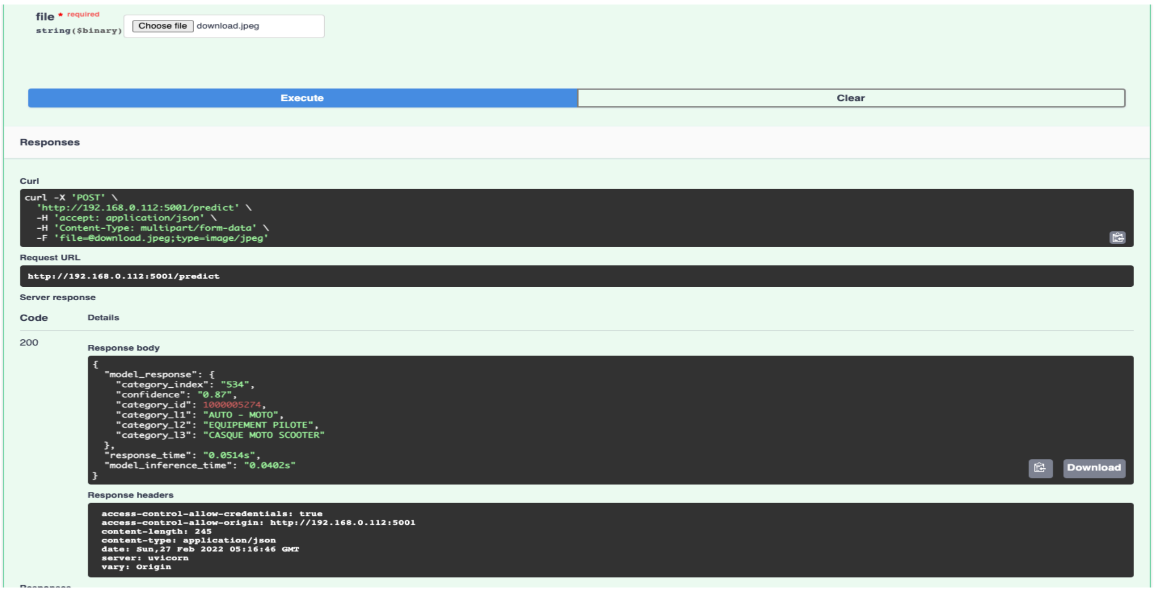
For using the API without streamlit you can use FastAPI endpoint itself.
Follow the URL mentioned below:

http:<ip_address>:<port_no>/docs/

In the docs route you can see the different API calls listed. Out of one is a POST request which allows you to post a image directly giving the response in a JSON format as shown below.



Response:



Curl

```
curl -X 'POST' \
  'http://192.168.0.112:5001/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@download.jpeg;type=image/jpeg'
```

Request URL

```
http://192.168.0.112:5001/predict
```

Server response

Code | Details

200

Response body

```
{
  "model_response": {
    "category_index": "534",
    "confidence": "0.87",
    "category_id": 1000005274,
    "category_l1": "AUTO - MOTO",
    "category_l2": "EQUIPEMENT PILOTE",
    "category_l3": "CASQUE MOTO SCOOTER"
  },
  "response_time": "0.0514s",
  "model_inference_time": "0.0402s"
}
```

Response headers

```
 access-control-allow-credentials: true
 access-control-allow-origin: http://192.168.0.112:5001
 content-length: 245
 content-type: application/json
 date: Sun,27 Feb 2022 05:16:46 GMT
 server: uvicorn
 vary: Origin
```

A brief working demo video is created and can be accessed through the below link.
Demo Link :
https://drive.google.com/file/d/1XE1ur6mQDZLe7w3fgTR5u1lHKoZNnIlp/view?usp=sharing

Instructions to run the code is present in the ReadMe markdown in Git repository. Currently the app is deployed on my local system. It is an on-demand based demo systems. The demo can be shown on a video call as hosting the API on GCP continuously is a bit costly because all my Free trail accounts are exhausted.

Chapter 8

Conclusion

Conclusion and Learnings :

1. The work is not yet completed in terms of reaching the current SOTA on the dataset and is in progress because of the heavy dataset size, so I was not able to run multiple explorations in a short amount of time.
2. Even though I have used my personal laptop which is having an RTX 3080 and Google Colab Pro which is P100 GPU the training time for each epoch stands around 4-6 hours for finetuning and 6-11 hours for training from scratch.
3. So it was highly impossible to run multiple iterations with hyperparameter tuning and for a good number of epochs.
4. The entire training iterations and exercises are logged using weights &biases using the default logger for metrics and custom logger for learning rate. W&B was really helpful in keeping track of experiments and multiple loggings.
5. The best accuracy and F1 we have achieved are around 60.1 and 63.4 respectively from Inception V3 and some results around 60 percent using Efficient Net B7.
6. ResNet is a very nice architecture for converging but accordingly what I observed it is good to train a ResNet from scratch on this dataset to achieve good converge but this was highly impossible with the computing power and time we have right now.
7. The current model finalised is Inception V3 which is giving good performance on the current test set.
8. Inference time is fair enough around 0.1 s to run on a CPU based system , can be optimised further by using more sophisticated deployment approaches or using model quantization especially trying to use Knowledge distillation or qunatization aware training if more resources are there for training.
9. Further deployment can be optimized using Model deployment techniques especially the Deployment frameworks such as Tensorflow Serving, TensorRT etc.

But finally to conclude Efficient Net B7 and Inception V3 were able to give around 60% accuracy and 63% F1 on the dataset. I’m still running some training iterations with Efficient Net and Inception V3 with learning rate strategies this might give a bump in the accuracy but this is still in exploration.

Some of the future explorations which can be performed on the dataset are:

1. Ensembling different models is a bit hard problem as we are with the limited infrastructure.
2. Training Efficient Net B7 or InceptionV3 completely from scratch and adding custom layers which will result in around 10-11 hours with a batch size of 128 on a single GPU. But if there is a Multi GPU infra we should be able to reach 65-70 percent of Accuracy with the same if trained for at least 10-15 epochs.]
3. Ensembling Text-based learning (Label Content) with Image-based learning and creating a Hybrid ensemble to boost the accuracy further.
4. Optimizing the deployment further with Model Optimization either when training or with Deployment frameworks.
5. Using Deployment frameworks to make a continuous serving model (Using TF Serving or TensorRT, ONNX serving etc.) thereby reducing the inference time further.

References

1. https://www.kaggle.com/c/cdiscount-image-classification-challenge/data
2. https://github.com/hamzafar/cdiscount-image-classification-challenge
3. https://techblog.cdiscount.com/cdiscount-image-dataset/
4. https://towardsdatascience.com/comprehensive-guide-on-multiclass-classification-metrics-af94cfb83fbd
5. https://towardsdatascience.com/image-classification-in-data-science-422855878d2a
6. https://towardsdatascience.com/main-challenges-in-image-classification-ba24dc78b558
7. https://www.kaggle.com/c/cdiscount-image-classification-challenge/overview/evaluation
8. https://docs.streamlit.io/
9. Inception V3 - https://arxiv.org/abs/1512.00567v3
10. Resnet 110 - https://arxiv.org/pdf/1512.03385.pdf
11. Efficient Net: https://arxiv.org/abs/1905.11946
12. Keras API docs - https://keras.io/api/
13. TF Keras - https://www.tensorflow.org/api_docs/python/tf/keras
14. FastAPI docs - https://fastapi.tiangolo.com/