Cdiscount's Image Classification Challenge

Anindita Nath
Manoj Pravakar Saha
Md. Mahmudulla Hassan
Mohammad Sujan Miah

Department of Computer Science
University of Texas at El Paso

December 12, 2017

This study is undertaken towards completion of the coursework for *C*S-5354 at the *University of Texas at El Paso*

Contents

1 Introduction

Cdiscount.com is the largest non-food e-commerce company in France. The range of products the company sells includes almost everything from TVs to trampolines and the list continues to grow rapidly each year. By the end of this year, Cdiscount.com will have over 30 million products up for sale which is up from 10 million products only 2 years ago. The task is to ensure that each of these products is well-classified. This is indeed challenging. Currently, Cdiscount.com applies machine learning algorithms to the text description of the products in order to automatically predict their category. However, these techniques are already up to their maximum potential and hence, the company decided to use data science techniques to images as a step towards next quantitative improvement. This is the motivation behind this project.

This is a Kaggle Competition sponsored by Cdiscount.com. The goal of this competition is to predict the category of a product based on its image(s).In this challenge, we are required to build a model that automatically classifies the products based on their images. For every product *_id* in the test set, the correct *category_id* should be predicted.

2 Data Set

The data set of this competition has been made available in the given link kaggle webiste. This is unique and characterized by superlative numbers in several ways:

(i) There are almost 9 million products , i.e., half of the current catalogue.

(ii) There are more than 15 million images at 180x180 resolution.

(iii) There are more than 5000 categories, 5276 to be precise.

The train and test sets are in .bson format. BSON, which is short for Binary JSON, is essentially a binary-encoded serialization of JSON-like documents.

(i) *train.bson*: This file is of size 58.2 GB. This contains a list of 7,069,896 dictionaries, one per product. Each dictionary contains a product id (key: *_id*), the category id of the product (key: *category_id*), and between 1-4 images, stored in a list (key: *imgs*). Each image list contains a single dictionary per image, which uses the format: *'picture': b'...binary string...'*. The binary string corresponds to a binary representation of the image in JPEG format.

(ii) *train_example.bson*: This file contains only the first 100 records of *train.bson* so that exploring the data could be started before downloading the entire set.

(iii) test.bson: This file is of size 14.5 GB. This contains a list of 1,768,182 products in the same format as train.bson, except there is no *category_id* included. The objective of the competition is to predict the correct *category_id* from the picture(s) of each product id *_id*.

(iv) *category_names.csv*- This shows the hierarchy of product classification. Each *category_id* has a corresponding level1, level2, and level3 name, in French. The *category_id* corresponds to the category tree down to its lowest level. All the absolutely necessary information is found in train.bson.

(v) *sample_submission.csv*: This was provided to show the correct format for submission. As per the requirements mentioned in the competition, for each *_id* in the test set, a *category_id* must be predicted. The submission file should contain a header and have the following format :

_id	*category_id*
2	1000000055
5	1000016018
6	100016055

This competition is evaluated on the categorization accuracy of the predictions. In other words, the competitors are judged on the percentage of products that is correctly classified. The competition rules recommended that the submission file should be zipped before uploading for scoring.

3 Data Exploration

(i) *Extreme multi-label classification problem.* We are required to classify above 5000 categories.

(ii) *Imbalanced data set.* The data set is highly disbalanced *refer fig. 1 and 2* . Thorough scrutiny of the dataset revealed that while some categories contained only 3 images for all products, some had more than 1000.

To visualize the data set, the following were generated through code :

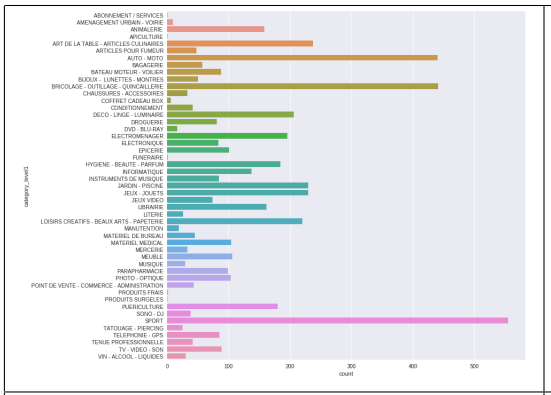

Fig.- 1: *Category Level 1*

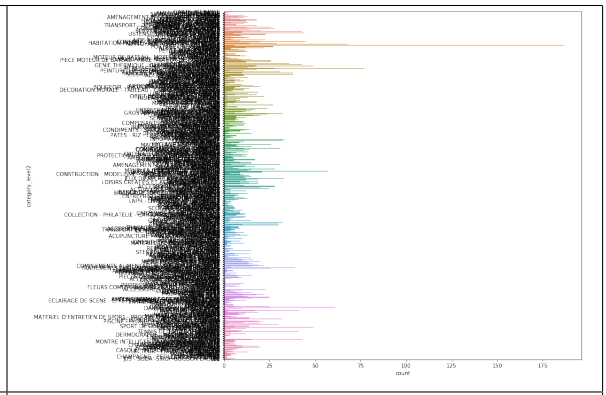

Fig. 2 *Category Level 2*

Visualization of Category Levels.

(iii) *Ambiguity in images.* The image list provided with each product was often not relevant. For better visualization, we extracted the images from the training set and conducted a naked eye inspection.

We observed that the images, at times, did not belong to the class they were paired with. Even humans fail to classify the products based on the given images. Some examples are shown below.

Apparently, there is no way one can deduce the class the image *Fig.3* or *Fig.6* belong to. Again, the image in *Fig.4* belonged to the class of mobile case. However, we could also see the image of a mobile phone merged with it.

Both the training and test data set is in .bson format. To read directly from the .bson files, we employed the data generator class in Keras, *'BSONIterator'*.

We, then, followed a standard 80-20 split of the training set to generate the validation set. These were saved in *train.csv* and *val.csv* files, respectively.

This was done only once. Next time, these csv files were only loaded and images were generated from them to be used for training and testing each model. This technique , thus, saved computing time and resource to a considerable extent.

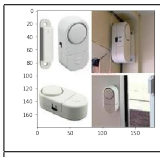

Fig. 3

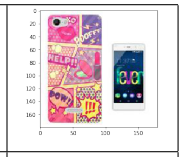

Fig. 4

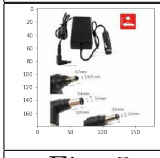

Fig. 5

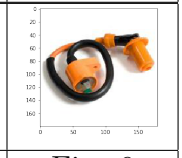

Fig. 6

Image set.

4 Experiment and Results

4.1 Pre-requisites

4.1.1 Computation Resources

Two different systems were used for computations in this project.

1. We have used a GPU-accelerated compute instance in AWS for training the networks. The compute instance was configured with a Nvidia Tesla K80 GPU. The compute instance was connected with a storage cluster that held the datasets, source code and trained models.

2. The second computing resource used for the project was configured with a Nvidia Titan XP GPU. Unlike the AWS instances, we had limited local access to this machine.

4.2 Model Construction and Optimization

(i) *Baseline:* Taking into account the size and complexity of the training set, it was necessary to apply an automatic feature extractor, considered better over hand feature engineering. Convolutional neural network is known to extract features automatically and better than a feed forward neural network. CNN's architecture is designed to take advantage of the 2D structure of an input image. The properties which help achieve this are local connections and tied weights followed by some form of pooling resulting in translation invariant features. Additionally, CNN is easier to train and has fewer parameters than fully connected layers with the same number of hidden units.

We implemented a simple convolution neural network as our baseline. It is comprised of 3 convolution layers with 32, 32 and 64 filters, respectively, and 1 fully connected layer. Max-pooling was used after each convolution. Number of channels was 3 (RGB) since the training set consisted of coloured images. We used categorical Cross Entropy as loss function. Training was done on 100 percent of training set. The details are tabulated in Table:1 - baseline.

However, validation accuracy obtained was as low as 0.0860 in 3 epochs. We did not run it for more epochs as the validation accuracy was only linearly increasing while the validation loss had already started to increase as well. Early stopping with patience=2 detected this. Training this simple convolution model took about 28 hours. The Amazon Web Services GPU instance *Nvidia Tesla K80* used cost $.90 per hour which limit our computation capacity.

(ii) *Downsize Image:* We figured the high resolution of the training images added to the complexity of training. Hence, in this model, we decreased the resolution of each image to 16 by 16, keeping the rest of the network parameters same as above. Also, the training was done on only 1% of the training data. The details are to be found in Table:2- downsize.

Training time, quite obviously, was reduced to 1 hour for 10 epochs and validation accuracy dramatically increased to 0.4189 from the baseline.

(iii) *Augmentation:*Along with the huge size of the training set, each of the category was highly imbalanced. Hence, batch-wise training on the above models ran the risk of over-fitting the data. To make the model more robust to over-fitting, image augmentation was used.

First, those categories with less than 3 images were dropped so that those categories with reasonable number of images remain. Then, each image attached with the product of the remaining category was augmented with its flipped, rotated and shifted version. Training was done on 1% of the set. Validation accuracy was found to be significantly higher than the baseline. Table:2- Augmentation and Fig.7-8 presents the details of the model.

(iv) *Hierarchical CNN:*The hierarchical convolutional neural network is inspired from the HD-CNN architecture proposed by [Yan+14]. The HD-CNN architecture implements a two-level category hierarchy - coarse category level and fine category level. The

architecture share some common layers which are used to learn the low level features. Then the coarse category classifiers are used to classify the easier to predict the classes, and multiple fine category classifiers are used to predict fine categories grouped under coarse categories. The probabilistic predictions from the classifiers are averaged to get the final predictions. The catch in this architecture is that the model has to learn the class hierarchy from the data.

Since we already had a three-level category hierarchy provided by Cdiscount.com, we propose a hierarchical architecture where a single coarse classifier is used to predict the 49 level 1 categories in the first pass. The predicted classes is then grouped into 49 different categories and fed into the same number of different fine category classifiers to predict the level 3 categories (the final class labels).

We have implemented the level 1 category classifier using a Inception v3 classifier. However, not all part of the level 2 hierarchy of the architecture was implemented, at the time of writing this report. Table:9- *Hierarchical CNN* and Fig. 15-16 lists the details of the model.

(v) *Inception V3:* Conventional "filters" could only learn linear functions of their inputs. The idea of complex 'filters' i.e. multi-layer perceptions were first introduced in *Network In Network* [LCY13]This leads to increased abstraction. Batch Normalization is included in each of the hidden layers. The idea is to go wider at the same time of going deep. The central idea is to spatially average the feature maps at the final layer and directly feed to the softmax classifier. This helps to learn more in less time. This implied less computational resource ensuring no loss of vital information. This is what came to be known as the Inception Algorithm. Several versions incorporating incremental changes were released over the subsequent years.

The following figures demonstrate Inception V1, V2 and V3 networks.

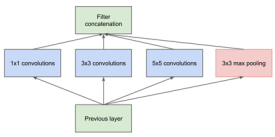

Fig.:InceptionV1

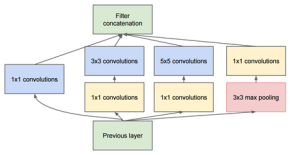

Fig.:InceptionV2

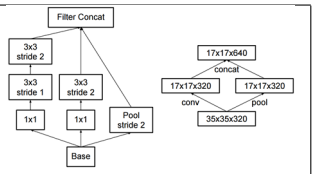

Fig.:InceptionV3

This algorithm was the central driving force behind our next successive models.

We started with Inception V3. The reason was i) it did exceptionally good on Image Net classification (top-5 validation accuracy was 93.9 *click on source*), and ii) it was already included in the Keras library.

Pre-trained weights on Image Net Classifiers were used to train the first 2 newly introduced layers of convolution, namely, Global Average Pooling and Dense Layer, for 1 epoch. Such initialization of weights was done to tune the model to match our number of prediction classes. Then, for the next 11 epochs, the entire network was allowed to

re-train the weights by itself. Weighted loss function was implemented to reduce the disadvantage of the unbalanced dataset.

Two models were built based on this algorithm. Both had 20% of training data but one of them had a batch size of 100 and the other of 400. The tabulated details are found in Table:4-*Inception V3_1*, Table:5- *Inception V3_2* and in Fig. 9-12. The highest top-5 validation accuracy was 0.7926.

(vi) *Inception V4:*In parallel with this, we implemented the Inception V4 [Sze+16] algorithm as well. However, this was not implemented in any kind of library. The code of Inception V4 had to integrated with our model from scratch. Major changes required were re-sizing each image to 299 by 299 as this was the default image size in Inception V4 code. Weighted loss function was similarly employed as above.

The model was first run for 3 epochs to check whether the validation accuracy was steadily increasing. Next, the model was run for 30 epochs. The details are tabulated in in Table:6-*Inception V4_1*, Table:7- *Inception V4_2* and in Fig. 13-14.

Top-5 validation accuracy increased to 0.4781. Though this was much less than that of Inception V3 but such direct comparison is not fair since here, training was done on only 1% of the entire train-set while Inception V3 was done on 20%.

However, further experiments with increased train size was not possible. GPU of the Amazon Web Services instance used crashed every time the size of the training data was increased by even slight percentage over 1% of the entire set. Also, the training time even for this small amount of train-data was about 23 hours.

(vii) *Inception ResNet V2:* As is observed above, though it had significant positive effect on the accuracy of the validation set, Inception V4 had huge computational expense. The paper, *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*[Sze+16] gives clear empirical evidence that training with residual connections accelerates the training of Inception networks significantly. There is also some evidence of residual Inception networks outperforming similarly expensive Inception networks without residual connections by a thin margin. This inspired us to implement the state-of-the-art model in Image Net Classification - Inception Resnet V2.

Training time was reduced by 2 hours. However, top-5 validation accuracy was almost close to that of Inception V4. There was no significant improvement.

However, the model was only run for 5 epochs. Though the accuracy was still increasing, we failed to run for more epochs since we ran out the allocated credits of $100 in our AWS account. We had no choice but to pause our experiments there.

4.2.1 Comparative Analysis

The following represents the comparative analysis of the models:

Network Parameters	Performance Metric	Remarks
3 Conv2D layers (32, 32 and 64 filters, respectively) DropOut Rate: 0.5 (after each MaxPooling layers and between 2 Dense layers) Trained on : entire set Batch Size : 128 Epochs : 3 Optimizer : Adam Learning Rate: 0.001 Loss function : Categorical Cross Entropy	Val-Acc : 0.0860	Training time: ∼28 hours Resource : Nvidia Tesla K80) Early stopping on val-loss introduced with patience=2

Table 1: Baseline

Network Parameters	Performance Metric	Remarks
Same layers of convolution as above but decreasing the resolution of each image to 16 by 16 Trained on : 1% data Batch Size : 128 Epochs : 10 Optimizer : Adam Learning Rate: 0.001 Loss function : Categorical Cross Entropy	Val-Acc : 0.4189	Training time: ∼1 hour Resource : Nvidia Tesla K80)

Table 2: Downsize Image

Network Parameters	Performance Metric	Remarks
2 CONV2D layer with augmentation DropOut Rate : 0.8 (after each MaxPooling layer and between 2 dense layers) Trained on : 1% data Batch Size : 128 Epochs : 5 Steps-per-epoch : 10 Optimizer : AdaDelta Learning Rate: 0.001 Loss function : Categorical Cross Entropy	Val-Acc : 0.3489	Training time: ∼4 hours Resource : Nvidia Tesla K80) Augmentation of train-images is done in those categories having more than 3 images. Flip, Rotate 90 degrees, shifting of the images applied

Table 3: Augmentation

Network Parameters	Performance Metric	Remarks
Inception -V3 Trained on : 20% data Batch Size : 100 Epochs : 11 Optimizer : Nesterov Adam Learning Rate: 0.001 Loss function : Categorical Cross Entropy	Top-1 Val-Acc :0.5509 Top-5 Val-Acc : 0.7926	Training time: ∼ 44 hours Resource : Nvidia Titan XP

Table 4: Inception V3_1

Network Parameters	Performance Metric	Remarks
Inception -V3 Trained on : 20% data Batch Size : 400 Epochs : 3 Optimizer : RMSProp Learning Rate: 0.001 Loss function : Categorical Cross Entropy	Top-5 Val-Acc : 0.5990	Training time: ∼ 16 hours Resource : Nvidia Titan XP

Table 5: Inception V3_2

Network Parameters	Performance Metric	Remarks
Inception -V4 Trained on : 1% data Batch Size : 100 Epochs : 3 Optimizer : RMSProp Learning Rate: 0.001 Loss function : Categorical Cross Entropy	Top-5 Val-Acc : 0.2119	Training time: $\sim$ 12 hours Resource : Nvidia Tesla K80

Table 6: Inception V4_1

Network Parameters	Performance Metric	Remarks
Inception -V4 Trained on : 1% data Batch Size : 100 Epochs : 30 Optimizer : RMSProp Learning Rate: 0.001 Loss function : Categorical Cross Entropy	Top-5 Val-Acc : 0.4781	Training time: $\sim$ 23 hours Resource : Nvidia Tesla K80

Table 7: Inception V4_2

Network Parameters	Performance Metric	Remarks
Inception Resnet Trained on : 1% data Batch Size : 100 Epochs : 5 Optimizer : RMSProp Learning Rate: 0.001 Loss function : Categorical Cross Entropy	Top-5 Val-Acc : 0.3521	Training time: $\sim$ 21 hours Resource : Nvidia Tesla K80

Table 8: Inception Resnet

Network Parameters	Performance Metric	Remarks
Trained on : 20% data Batch Size : 100 Epochs : 7 Optimizer : Adam	 Top-1 val-acc : 0.6151 (highest) Top-5 val-acc : 0.8325 (highest)	Resource : Nvidia Titan XP The accuracy decreased after reaching the highest point

Table 9: Hierarchical CNN

4.2.2 Graphical analysis

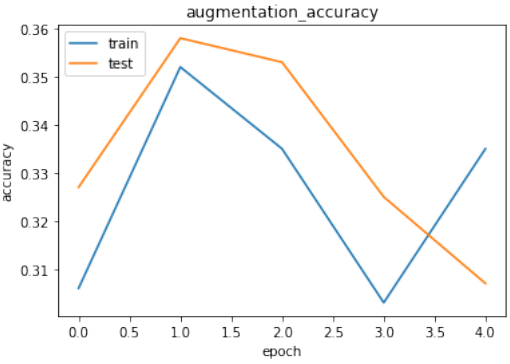

Fig.7: Augmentation_Accuracy

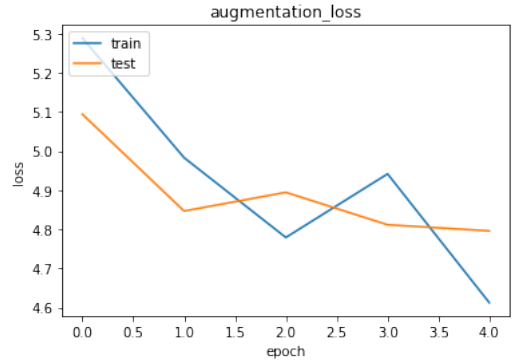

Fig.8: Augmentation_Loss

Fig.9: InceptionV3_1_Accuracy

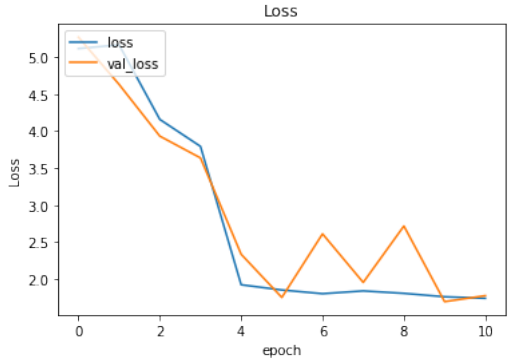

Fig.10: InceptionV3_1_Loss

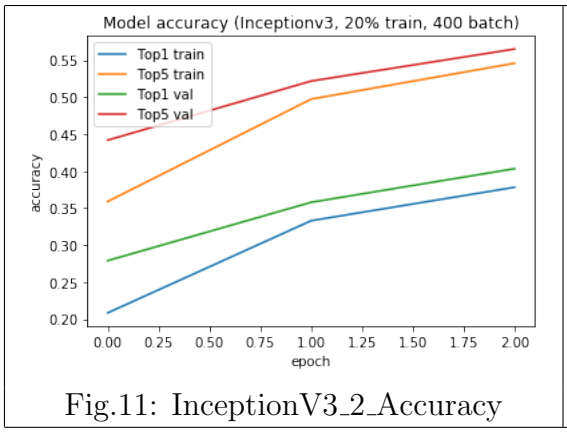

Fig.11: InceptionV3_2_Accuracy

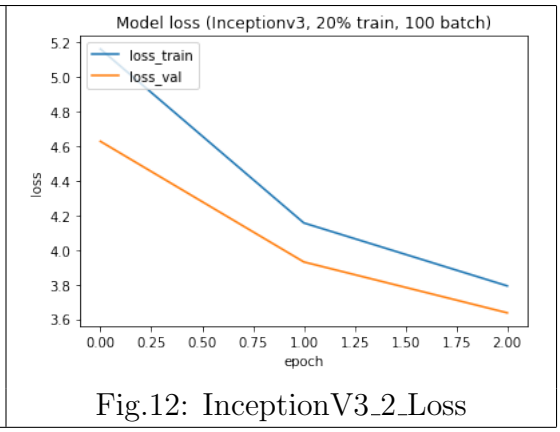

Fig.12: InceptionV3_2_Loss

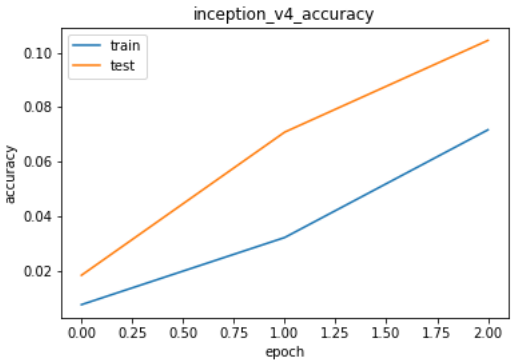

Fig.13: InceptionV4_Accuracy

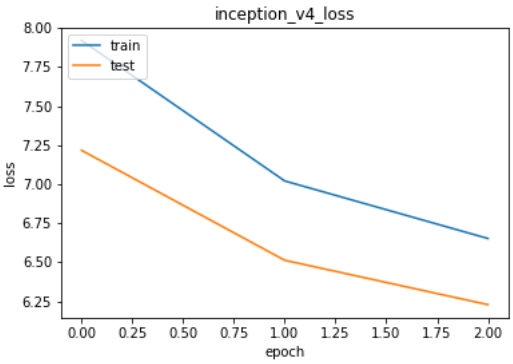

Fig.14: InceptionV4_Loss

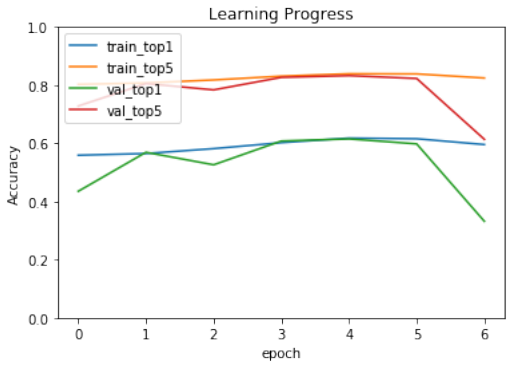

Fig.15: Hierarchical CNN Level 1 _Accuracy

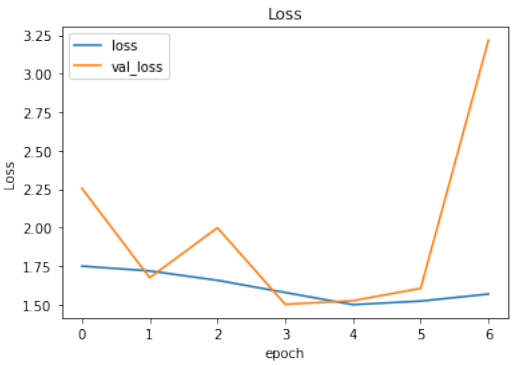

Fig.16: Hierarchical CNN Level 1 _Loss

5 Evaluation

5.1 Test Set Predictions

Inception V3 algorithm produced the highest top-5 validation accuracy as shown above. Hence, we have generated predictions for the test set using the Inception V3 model trained on 20% (effectively 16% of the data) of the data, with batch size of 100. These predictions were uploaded on Kaggle, and the website's verification service tested the predictions on 30% of the test set and produced the accuracy values for the models. Three different methods were used to generate the predictions.

(i) The first method sequentially fetches images for each product in the test set, generate prediction labels for each image, and then the predicted probabilities for all the images were averaged to calculate the final predictions. These predictions produced 42.80% accuracy.

(ii) In the second method, images for each product were fetched sequentially, and we considered the maximum of the predicted probability values as the final prediction. These method produced an accuracy of 42.40%.

(iii) The third method used only the first image of each product to generate the predictions. The accuracy obtained through this method is 41.25%.

From these results, it seems that averaging the predicted probabilities produces the best results. However, it should also be noted that these accuracy values are based on only 30% of the test set.

Our highest rank achieved in the competition leaderboard was 259 (when the team count was 504). The current rank is 304 (the team count has increased to 612). [1]

6 Conclusion

This project was a challenging one for us. We encountered a lot of hurdles, and each one of them was an opportunity for us to learn and increase our skills. We have learned the following things:

The training set was very large and unbalanced. We didn't have prior knowledge on how to handle such a dataset. So we had to find alternate ways to train our networks. To alleviate the problem of having an unbalanced dataset, we have used the weighted loss function that helped to achieve a balanced distribution of losses over all the classes. Handling a large dataset was also very challenging since we had to maintain a large storage system and find way to access the images quickly while training the network. After trying several techniques we've found that the built-in data generator in Keras does a very good job in this type of scenario. Combining this generator with the image preprocessing library in Keras which generates augmented data on the fly was helpful.

Since the dataset has categorical labels, we took the benefit of it. Instead of predicting the end-label, we modified our network in such a way that it predicts the first major class

[1] Kaggle statistics

and continues doing that until the last level. This type of hierarchical prediction improves the prediction performance of the model (Zhicheng Yan et. al. ICCV 2015).

7 References

[Sze+16] Christian Szegedy et al. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *ICLR 2016 Workshop*. 2016. URL: https://arxiv.org/abs/1602.07261.

[Yan+14] Zhicheng Yan et al. "HD-CNN: Hierarchical Deep Convolutional Neural Network for Image Classification". In: *CoRR* abs/1410.0736 (2014). arXiv: 1410.0736. URL: http://arxiv.org/abs/1410.0736.

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: *CoRR* abs/1312.4400 (2013). arXiv: 1312.4400. URL: http://arxiv.org/abs/1312.4400.