

CDiscount Kaggle Competition

Team: whatever

Yuya Ong, Tejas Shahpuri, Naveen Muthumanickam

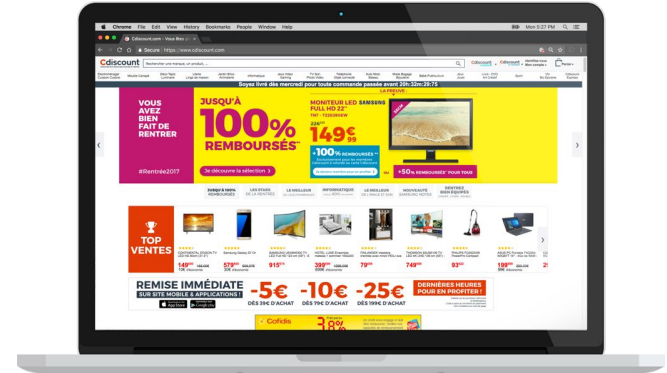
DS 310: Machine Learning Kaggle Competition Project

Outline

- Challenge Objective
- The Dataset
- Model Architectures and Training Strategy
- Results
- Lessons Learned

CDiscount

- The largest e-commerce site in France, very similar to Amazon.
- Generated up to 3 billion euros in the previous year in revenue from their platform.
- Their platform is expected to have up to 30 million different products that consumers can purchase from their platform.



Competition Objective

- CDiscount uses machine learning to classify their product categories - mostly relying on linguistic features (most probably word embedding features).
- CDiscount now wants to exploit image features from their product database to improve their classification of product categories.

Objective: Given images from their product database, predict the category of products.

The problem breaks down to a multiclass classification problem of using **images as inputs**.

Optimization Objective

The competition is evaluated based on the categorization accuracy of the predictions our model makes for each of the products.

Essentially this would just maximizing the percentage of the products we get correct.

Dataset Files

CDiscount provided the following files, each of which were a dump file from a MongoDB Database dump file.

- **train.bson**: List of 7 million dictionary entities containing a product id, 4-5 pictures of the product and the category ID.
- **test.bson**: List of 1 million dictionaries containing just the product id and the 4-5 pictures without any of the category ID.
- **category_names.csv**: For each category, there is a hierarchy of categories which are divided into three different levels: Level 1, Level 2, and Level 3.

Phase 1: Provides only 30% of their dataset.

Phase 2: Will be scoring it based on the 70% remaining set of the data.

Dataset Samples

1000010653
TELEPHONIE - GPS
ACCESSOIRE TELEPHONE
COQUE TELEPHONE - BUMPER



1000004079
INFORMATIQUE
CONNECTIQUE - ALIMENTATIO
CHARGEUR - ADAPTATEUR SEC



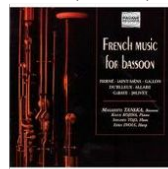
1000004141
INFORMATIQUE
PROTECTION - PERSONNALISA
COQUE - HOUSSE



1000015539
BRICOLAGE - OUTILLAGE - Q
SECURITE MAISON
ALARME AUTONOME



1000018290
MUSIQUE
CD
CD MUSIQUE CLASSIQUE



1000010653
TELEPHONIE - GPS
ACCESSOIRE TELEPHONE
COQUE TELEPHONE - BUMPER



1000018306
MUSIQUE
CD
CD VARIETE INTERNATIONALE



1000010961
TV - VIDEO - SON
CASQUE - MICROPHONE - DIC
CASQUE - ECOUTEUR - OREIL



1000010653
TELEPHONIE - GPS
ACCESSOIRE TELEPHONE
COQUE TELEPHONE - BUMPER



1000005744
AUTO - MOTO
PIECES
BOBINE D'ALLUMAGE - BOBIN



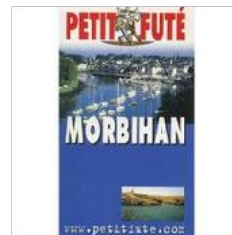
1000004079
INFORMATIQUE
CONNECTIQUE - ALIMENTATIO
CHARGEUR - ADAPTATEUR SEC



1000010667
TELEPHONIE - GPS
ACCESSOIRE TELEPHONE
HOUSSE - ETUI - CHAUSSETT



1000015309
LIBRAIRIE
AUTRES LIVRES
AUTRES LIVRES



1000010653
TELEPHONIE - GPS
ACCESSOIRE TELEPHONE
COQUE TELEPHONE - BUMPER



1000007361
BRICOLAGE - OUTILLAGE - Q
PEINTURE - REVETEMENT MUR
PEINTURE - VERNIS - TRAIT



Challenges

Although the competition itself seems pretty easy at the surface, here is where things gets really difficult and challenging:

1. **Dataset Scale:** The dataset contains around 9 million products, which amounts to **15 million images** at 180 x 180 resolution. (~60 GB compressed, 0.5 TB uncompressed).
2. **Class Size:** The competition requires us to classify **at least 5000+ different classes** of products.

To put this into perspective...

The current state of the art models are based on the **ImageNet Dataset**, which only has around 1.2 million images and 1000 classes to classify.

Hence, this challenge not only becomes difficult to model but also to engineer to have it scale against such size.

Challenges of Modeling

One of the largest challenges in the competition is trying to scale the training process:

- Loading the dataset using various batch size parameters - note disk I/O speeds are fairly slow and is a huge bottleneck in our data pipeline.
- Learning large number of parameters/weights efficiently.
- Account for huge variability in the dataset.
 - Images contain many different perspectives.
 - Some images are composites of many images.
- Training a model will take days due to immense number of data and parameters - slow development/iteration cycles for modeling, even with GPU.

Managing Scale

As our dataset is large, the methodology for training on our dataset changes significantly:

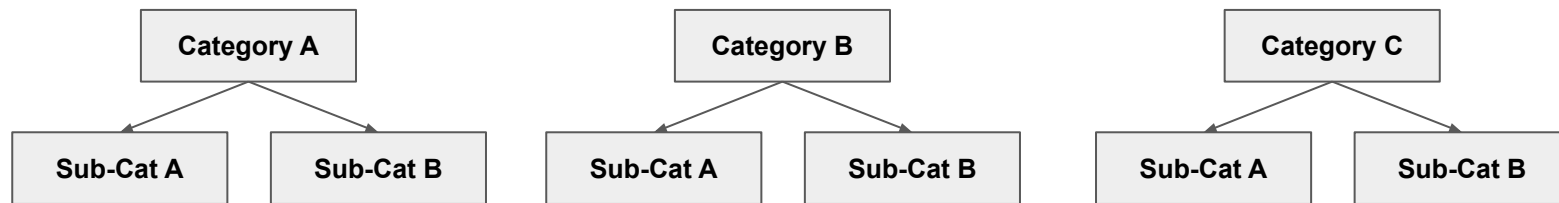
1. Can't fit all of the data on the memory in one shot while training on the dataset.
2. Very hard and computationally expensive to perform K-Fold Cross Validation.

In short, we will have to **make some sacrifices** in order for us to manage the scale complexity of the data with the cost of jeopardizing accuracy (and execution time) of our model.

Modeling Strategy 1: Exploitation of Hierarchical Structure of Labels

Strategy: Train the model by exploiting a tree based structure of the category - assuming that majority of the hierarchy's leaf nodes are independent from one another.

Key Idea: This reduces the number of parameters we train for and improve the overall precision of the model itself. Within each level, we can train the model by pruning it based on specific categories.



Problem: This would lead to many small models, which increases overall complexity of the modeling and development process... *Ultimately did not really work out.*

Modeling Strategy 2A: Pre-Trained Models

Strategy: Train a model from existing pre-trained weights and prune the model based on the given dataset to expedite the overall training process.

Key Idea: When training models, we would normally set the weights to some random value and then attempt to search for the most optimal values. In this case, one of the most used models is based on the ImageNet dataset.

Models Implemented:

- Inception V3*
- Xception V3

** Due to time constraints, we were only able to fully train one model, but we also have another model which we plan to further train in the future.*

Modeling Strategy 2B: An Ensemble of Pre-Trained Models

Strategy: In addition to training different models, we can combine the predictive results across several different model to increase the predictive power of the Deep Learning models. We can achieve this by ultimately introducing an ensemble.

We can utilize an averaging strategy to utilize the average probability across two of the models for the inference process.

Main Idea: As our models were trained under different hyperparameters and different subsets of the data, we believe that using this strategy will help to account for the variability factor of the dataset.

Models to Combine in our Ensemble:

- Inception V3
- Xception V3

Modeling Strategy 3: Data Augmentation

Strategy: Since the dataset is quite imbalanced, we can improve the overall accuracy of the model by ensuring we introduce variability in our dataset.

Since we are working with image datasets, we can simply do this by taking an image and introducing random perturbations through various transformations.

Data Augmentation Types

- Rotation
- Shearing
- Reflection
- Translation
- Image Cropping
- Rescaling



Modeling Strategy 4: Shifting Probabilistic Interpretation of Models

When building the models we can formulate two different probabilistic interpretations.

This would potentially improve the way we can better set our objective function since the competition considers the percentage of products (which comprises of several images together).

Considering Each Product Image Separately

Given an image x , we predict a probability $p = \Pr(Y = y \mid x)$, where Y is the category and $y = \{1, \dots, 5270\}$.

So, given a product T , where $T = \{x_1, x_2, \dots, x_n\}$, and n is the number of photos given to us within T , we can consider the average probability q , where $q = (p_1 + p_2 + \dots + p_n) / N$.

Modeling Strategy 4: Shifting Probabilistic Interpretation of Models

Considering Each Product Image Jointly

Given a product T , where $T = \{x_1, x_2, \dots, x_n\}$, and n is the number of photos given to us within T ...

We can try to *jointly model* each of the images, where we directly predict q as:

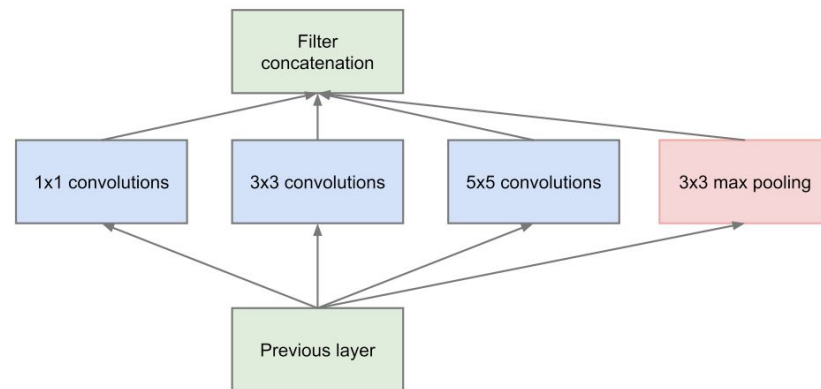
$$q = \Pr(Y = y \mid x_1, x_2, \dots, x_n).$$

This would be potentially better, since if we tried working with averages, then one bad image prediction can significantly reduce the score down.

We would have to come up with some function, *not using averages*, to estimate this probability using the individual probabilities of $\Pr(Y=y \mid x_i)$ from each image in the set of T .

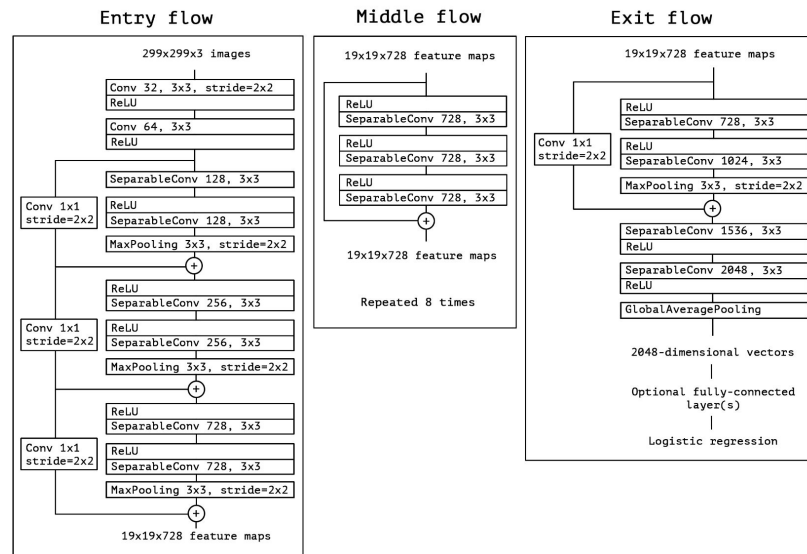
Model A: Inception V3

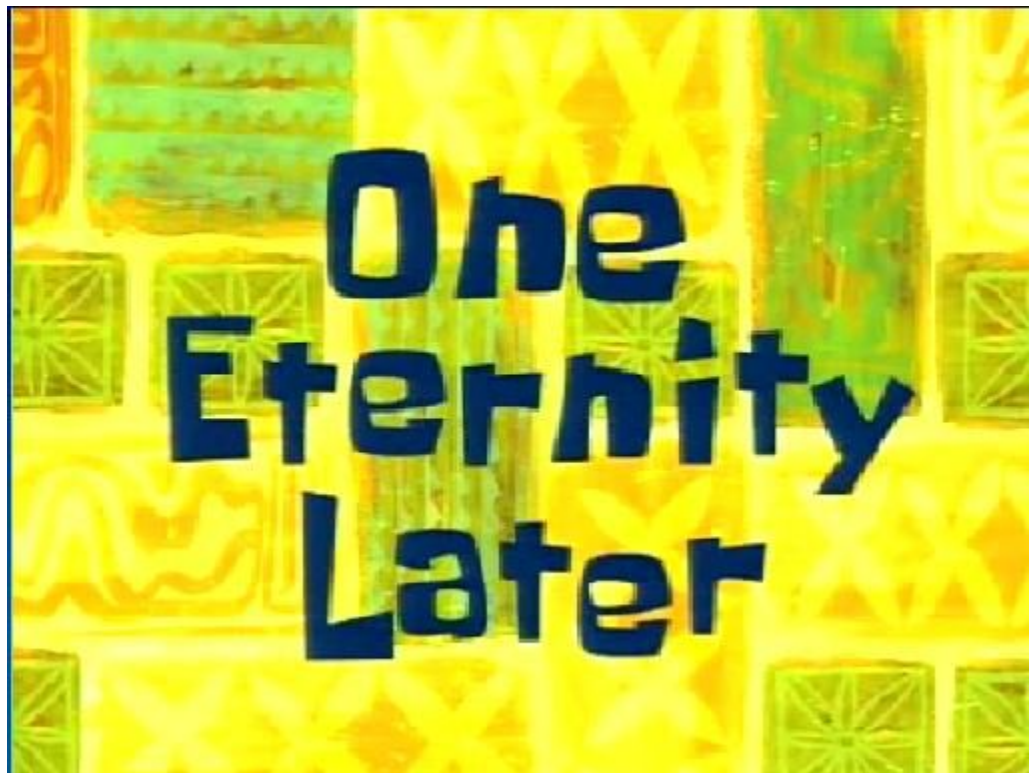
- Model developed by Szegedy et. al from Google Brain's Research Team.
- Conventional CNN filters only try to learn linear functions - the Inception architecture tries to learn a nonlinear function through a MLP.
- Reduces the number of FC layers on top by introducing the idea of **Global Average Pooling** - they attempt to spatially average the feature maps at the final layers, thus reducing overfitting and removing the FC layer at the end, which speeds up training.



Model B: Xception Architecture

- Xception = Extreme Inception Model
- Improvement over the Inception V3, proposed by Francois Chollet (author of Keras).
- Models based on the hypothesis that mapping of cross-channel correlations and spatial correlations in the feature maps of CNNs can be entirely independent.
- Essentially a stronger version of the Inception Architecture.
- Utilizes **Depthwise Separable Convolution Operations**.
- Basically a Depthwise Convolution followed by a Pointwise Convolution
- Performance gain due to efficient use of the model parameters.

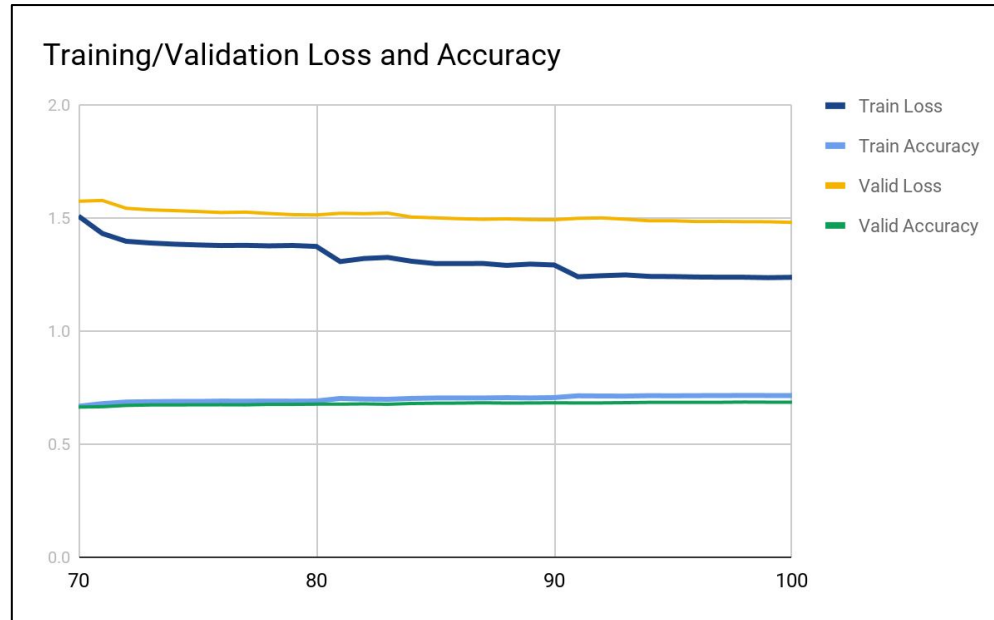





























Sorry for hogging the GPU cluster for 5+ days...

Inception V3 - Loss and Accuracy Graph

The following graph show our training and validation loss and accuracy rates from the 70th epoch.

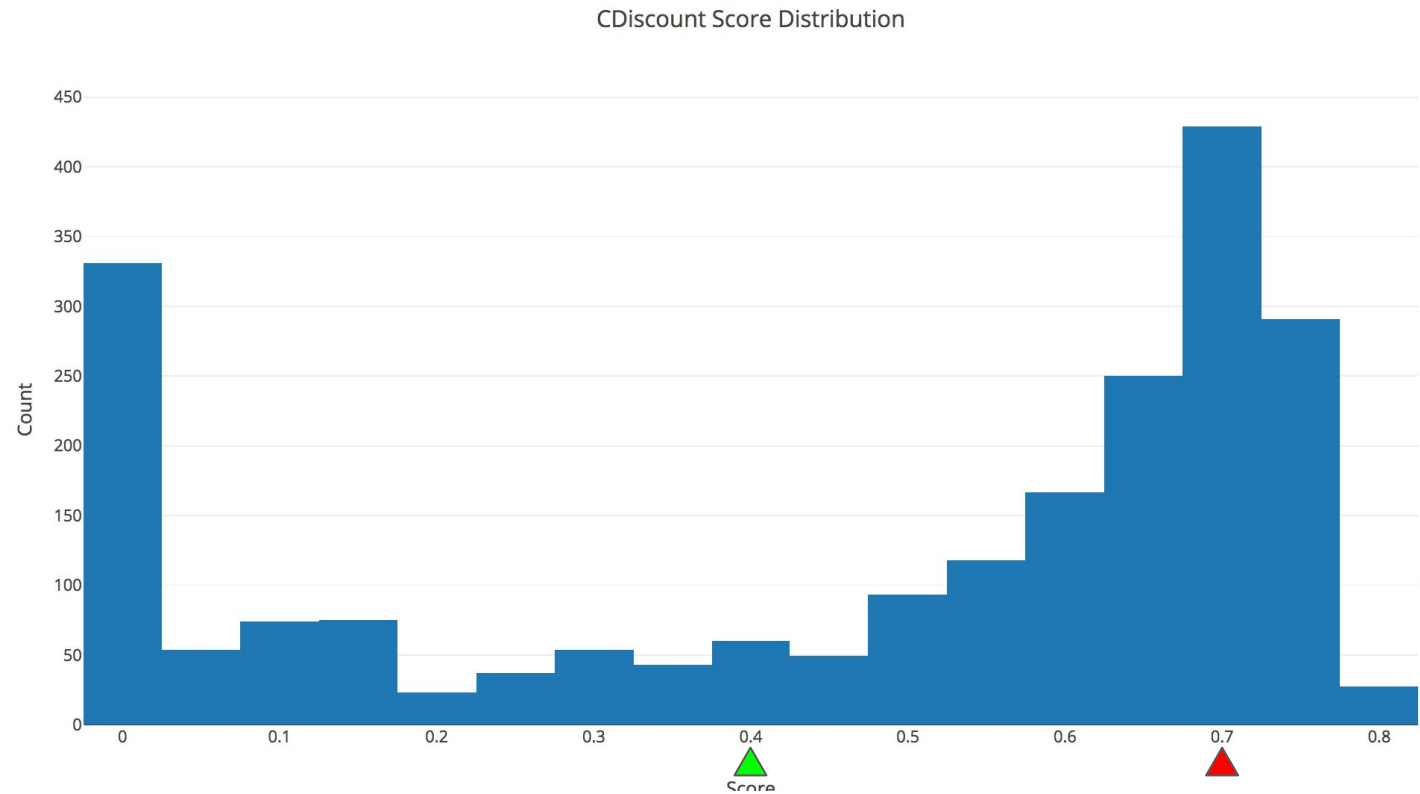


As of December 6th, 2017 1:51 PM...

Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions
87	new	RossWightman					 0.70324	3 3h
88	▼ 13	Survivors				  +4	0.70317	10 12d
89	▼ 13	Fengjinghao					0.70307	2 2d
90	new	chicken dinner					0.70296	5 2d
91	▼ 13	Fast Science				 	0.70289	3 1mo
92	▼ 13	ODS Minsk				 	0.70269	15 7d
93	▼ 13	DavidGbodiOdaibo					0.70244	1 10d
94	▼ 13	Beri					0.70225	3 1mo
95	▲ 228	whatev				  	0.70146	6 1d
Your Best Entry ↑								
Your submission scored 0.70146, which is an improvement of your previous score of 0.16718. Great job!							 Tweet this!	
96	▼ 14	MA					0.70114	25 5d
97	new	Null					0.70027	2 1d
98	▼ 14	out of gpus					0.69973	3 2mo
99	▼ 14	WispZero					0.69934	6 1mo
100	▼ 13	amaia					0.69808	2 2mo
101	▼ 13	Aurora@*					0.69726	16 1mo
102	▼ 13	Zshiney					0.69697	5 1mo
103	▼ 9	MLFriends				  	0.69614	28 5h

Our team has struck top 100 (top 16% ranking wise)!

Benchmark of Score Distribution



Kaggle Submission Results History

Date	Model	LB Score
11/04/2017	Sample Baseline	0.00868
11/11/2017	CNN Model 1	0.16718
11/20/2017	CNN Model 2	0.00892
11/20/2017	CNN Model 3	0.00868
11/20/2017	Inception V3 Prototype Model	0.02502
12/4/2017	Inception V3 Final	0.70146

Lessons Learned

1. Pre-Trained weights can make a huge difference in how fast you are able to build models - you can significantly save lots of time using existing weight parameters.
2. Hyperparameter selection is key - especially batch size and learning rate as they make a huge difference in how your model will find the most optimal parameters.
3. Hardware constraints and obstacles don't necessarily have to stop you - sometimes it gives an opportunity for you to provide much more creative and scalable solutions to your problems.

Questions?