Domain Background

This is a Kaggle challenge provided by Cdiscount.com. Cdiscount.com is the largest non-food e-commerce company in France. It generated nearly 3 billion euros last year. While the company already sells everything from TVs to trampolines, the list of products is still rapidly growing. By the end of this year, Cdiscount.com will have over 30 million products up for sale. This is up from 10 million products only 2 years ago.

Problem Statement

With such a wide ranging number of products, it's important that they are classified properly so that potential customers can find what they are looking for. However, ensuring that so many products are well classified is a challenging task. Manual classification is out of the question. Relying on the seller to post accurate product classification could work, but they are incentivized to game the system in order to get more traffic to listing.

Currently, Cdiscount.com applies machine learning algorithms to the text description of the products in order to automatically predict their category. This is working well, but these methods now seem close to their maximum potential.

Cdiscount need a novel way to continue to improve the accuracy of their classification methodology. The one area they have not explored is to use the images in each listing for classification.

Datasets and Inputs

The data set Cdiscount.com is making can be characterize by the following:

- Almost 9 million products: half of the current catalogue
- More than 15 million images at 180x180 resolution
- More than 5000 categories

Below is a description of the input file

- train.bson - (Size: 58.2 GB) 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. This kernel provides an example of how to process the data.
- train_example.bson - Contains the first 100 records of train.bson
- test.bson - (Size: 14.5 GB) 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). The category_ids that are present in Private Test split are also all present in the Public Test split.

- category_names.csv - 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. This hierarchical data may be useful, but it is not necessary for building models and making predictions. All the absolutely necessary information is found in train.bson.
- sample_submission.csv - Shows the correct format for submission.

Solution Statement

Create an algorithm that automatically predict the category of a product purely based on its image(s). Note that a product can have one or several images associated. For every product *_id* in the test set, it should predict the correct *category_id*.

Benchmark Model

The current benchmark model used for image classification problems is Convolutional Neural Networks (CNN).The way CNN work is that instead of feeding the entire image as an array of numbers, the image is broken up into a number of tiles, the machine then tries to predict what each tile is. Finally, the computer tries to predict what's in the picture based on the prediction of all the tiles. This allows the computer to parallelize the operations and detect the object regardless of where it is located in the image. There are already CNN that researchers have built and achieved very impressive results. The more famous ones: VGG19, Resnet50, Inception and Xception. Using those models on an image library of dogs, we can able to obtain accurate classification more than 80% of the time. We should be able to take a similar approach here and get similar results. Currently, the leader of the competition achieved an accuracy of 77% on the test dataset.

Evaluation Metrics

The evaluation metric will be the categorization accuracy of the model's predictions (the percentage of products it gets correct).

For each *_id* in the test set, it must predict a *category_id*. The file should contain a header and have the following format:

```
_id,category_id

2,1000000055

5,1000016018

6,1000016055

etc.
```

Project Design

I think generally, the approach I will take is to build my model around the existing pre-trained CNN and tune it appropriately. There are many people way smarter than me researching this problem, there is no reason why I can't leverage what they have already built and apply it to this problem.

I plan to approach the problem in the following 3 stages.

1. Data exploration and understanding
2. Model architecture exploration
3. Model tuning

Data exploration

In this phase, my goal is to simply explore the data to get a better understanding of its content and structure. Some of the questions that I would want answers to off the bat are:

- What are the categories? How specific do they get?
- How are the images represented? Do I have to do pre-processing on them?
- On average, how many images are there per listing? What are those images like (different view of the same product? Lifestyle shots of product in action? Close up shots of products? etc.) I can see different type of shots can confuse and vastly skew the result one way or another.

Model architecture exploration

Once I have process the data, now it's time to explore how well the current state of the art CNN perform on the dataset. I will connect some of the well known CNNs with a final Dense layer with softmax activation function and evaluate their accuracy. The CNNs I plan to try are:

- Resnet50
- VGG19
- InceptionV3
- InceptionResnetV2

Another thing I would like to explore is whether one algorithm tend to perform better in one broad category than another. For example, maybe Resnet50 perform better with phones, but InceptionV3 perform better on desks. If so, there may be value in identifying the broad category first and then use a separate CNN for the specific category classification.

Model tuning

Once I figure out the basic model architecture, the next step is to tune the model in order to optimize the weights while ensuring it's not overfitting. A few levers I can use here are:

- Image augmentation
- Number of Epochs
- Add additional fully connected layers after CNN