CAPSTONE PROJECT RAJ DAYAL NATARAJAN

MACHINE LEARNING NANODEGREE DATE: JAN 22nd, 2018

DEFINITION:

Project Overview:

The problem I chose to tackle for my project is a Kaggle competition which ended on December 15th. The project name is “Cdiscount’s Image Classification Challenge”. The following is the background. Cdiscount.com is France’s largest non-food e-commerce company with a revenue of 3 Billion Euros. The company sells everything from TV’s to trampolines and has about 30 million products which are up for sale by the end of the year. The company now faces a challenging problem of ensuring that all these product are correctly classified so that the website can be streamlined to provide a good user experience to the customers.

Problem Statement:

The problem seen by Cdiscount as mentioned in the background is as follows. With an ever growing array of products being sold by Cdiscount, the company faces the challenge of correctly classifying these products into distinct categories so as to improve the customer’s online experience with Cdiscount.com.

Currently, Cdiscount is using Machine Learning techniques to infer the classification of the product from the description of the product. However, the company thinks that the next major quantitative improvement in classification accuracy can only come from using images to classify the products in addition to the description.

Therefore the goal is to create an image recognition algorithm to classify products correctly using a suitable metric and compare it with a baseline benchmark to evaluate the performance. The following is the break up of all the tasks involved.

1. Download the dataset and extract the images from the compressed data set provided
2. Subset the problem to be tackled - since this is a big data challenge, I will be sizing the problem down from the original problem set on Kaggle
3. Complete the necessary preprocessing and preliminary exploratory data analysis
4. Start building the classifier and evaluate performance against a baseline model

Metrics:

I will be using Categorization accuracy as the evaluation metric for this data set - the percentage of the products identified, which are correct. This metric is relevant for an image labeling problem and is also the metric for the Kaggle competition. Hence I will be using Categorization accuracy as the evaluation metric.

Categorization Accuracy = Total correct identifications / Total predictions given

The model is a CNN with an output softmax layer, which gives an output probability for each of the 49 categories. Instead of using a metric like precision or recall, it is better to use accuracy for this challenge. With precision or recall, the emphasis is more on the accuracy of the positive label predictions(Tp / Tp+Fp) and the percentage recognition of positive categories (Tp / Tp+Fn). However, in this case, all labels are equally valid and important, hence it might be enough to use Categorization accuracy.

Tp - True Positive, Fp - False Positive, Tn - True Negative, Fn - False Negative

ANALYSIS:

Data Exploration:

Data sets and Inputs:

The data set has been provided by CDiscount is from its internal catalogs and has about 9 million products with 15 million images with a standard resolution of 180x180. There are approximately 5200 output categories. In short, it would be a big data, multi-class classification problem.

The data set has been provided in BSON format - binary encoded serialization of JSON like documents. On exploring the data set, I found that the training data set has ~5200 categories and around 9000 images overall belonging to all these categories. There is also an asymmetrical distribution of these images across each of the categories. Therefore, for some of the categories, there will be not be enough images to train. To compensate for this, some data augmentation is required. I will be using the ImageDataGenerator from the Keras preprocessing module to do the data augmentation for the sparsely filled categories. Available images will be augmented – rotation, shear, zoom etc. But in addition to this, the augmentation also has to include images of different colors and gray scale images for each of the product categories.

However, after discussion with a Udacity coach (after running into issues with memory requirements even on AWS), I decided to reduce the scope of the problem. The idea would be to take 20-50 product categories, augment the images in the data set to produce a rich set of images. These will be divided into validation and test sets and will be used to evaluate the model.

Details: (Note: From the competition description)

Training Data set:

" *(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."*

Testing Data Set:

"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."

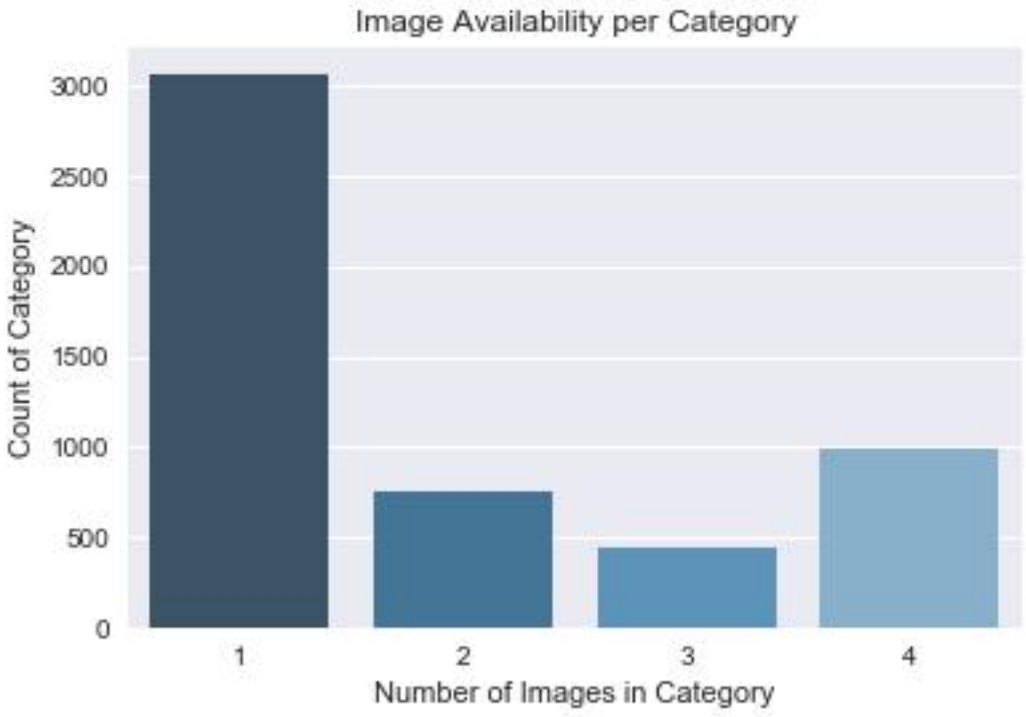
Category names:

"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."

Exploratory Visualization:

As can be seen, the number of categories provided for this data set is very large ~5200. Since I have chosen to reduce the scope of the problem in lieu of resource limitations, I will select the categories to be used based on the number of images available for each category.

```
Counter({1: 3065, 4: 999, 2: 756, 3: 450})
I had used a counter function to generate the count of categories having 1,2,3,4 images (Note the
maximum number of images per category is 4).
```



On exploring the training data set provided, the images per category is not very high. The following is a break down of how the categories stack up in terms of images available. The breakdown suggests that there are onl y 999 categories which have 4 images. Therefore, I have chosen to use 50 of these 999 categories to build m y classifier. I will use image augmentation to increase the sample size of each category selected. Stats and vi sualization shown above.

Algorithms and Techniques:

I will be using Convolutional Neural Networks to solve the Image recognition challenge. In this section, I will try to summarize my understanding and reasoning for using Convolutional Neural Nets for this particular problem.

Convolutional neural nets came to the fore in the early 2000's having produced significant improvements in the image recognition sphere. With the widespread usage, the performance has only improved over the years becoming the defacto algorithm for image recognition.

Understanding Convolutional Neural Networks:

Architecture – Describes the data flow architecture - from the Input Image to the output classification labels. In addition, I will describe the function of each of the layers. I will try to illustrate my understanding of the CovNets with a simple example as well.

Architecture:

A convolutional neural network is multi-layer network containing an input layer, an output layer and a number of convolutional layers and max pooling layers. The number of layers in a CNN is not definite and the general idea is that – As the number of convolutional layers increase, it results in the production of a more complex feature set which gets fed into the final output classification layer.

Input:

A computer processes an image as grid of pixels, with 3 channels (RGB). So for example, a picture which is 48 x 48 pixels and is colored, will be represented by an array of numbers of dimensions – 48 x 48 x 3. The 3rd dimension represents the intensity of the pixel for that color (represented by each channel or array).

So the three dimensions can be summarized as follows:

1. Number of pixels along the X direction of the picture: 48 pixels
2. Number of pixels along the Y direction of the picture: 48 pixels
3. Number of channels at each location or pixel – 3 (One corresponding to the intensity for Red, Green and Blue)

Convolutional layer:

Now that we have defined the input – let us start describing the first layer or the input layer of a CNN which is always a convolution layer. The convolutional layer in essence functions as a feature extractor at each stage. It is named so because of the operation it performs on the inputs being fed into it – Convolution between the input (Input array from the image in this case – 48x48x3 and the filters which have been designed by the developer). A convolution layer preserves the spatial relationship between pixels by performing convolutions in small patches or grids. i.e. the operation is performed on adjacent pixels of the image.

Filter:

A filter also known as Kernel is a tailored matrix of numbers which are designed to capture a specific pattern in the image. When the input image is convolved with the filter or the Kernel by the convolution layer, the output of the convolution detects whether the pattern in the filter is present in the image. Therefore the filter can be thought of a feature detector.

This is best explained by a simple example. The below is the most granular version of how a convolution will take place. Once we understand this we can extrapolate this to how a convolution layer works. In the below figure, let us assume that the stick man is the input image. The right side represents how this will be seen by a computer as intensity values for pixel points. (This case shows a single channel for demonstration). The filter in this case is a vertical edge filter which is slid across the entire image with steps of one cell each time from left to right. The output is convolved image values.

Let us take the first convolution operation to see how the math works here.

Since the filter (Row x Column) is a 3x2 matrix, the first operation will take the first 3x2 portion from the image highlighted in brown and matrix multiplied to produce the value 0. Then the filter is slid across the image (Red portion) and the operation is repeated. This process is repeated across the image to come up with the convolved image

Op1: Output - 0

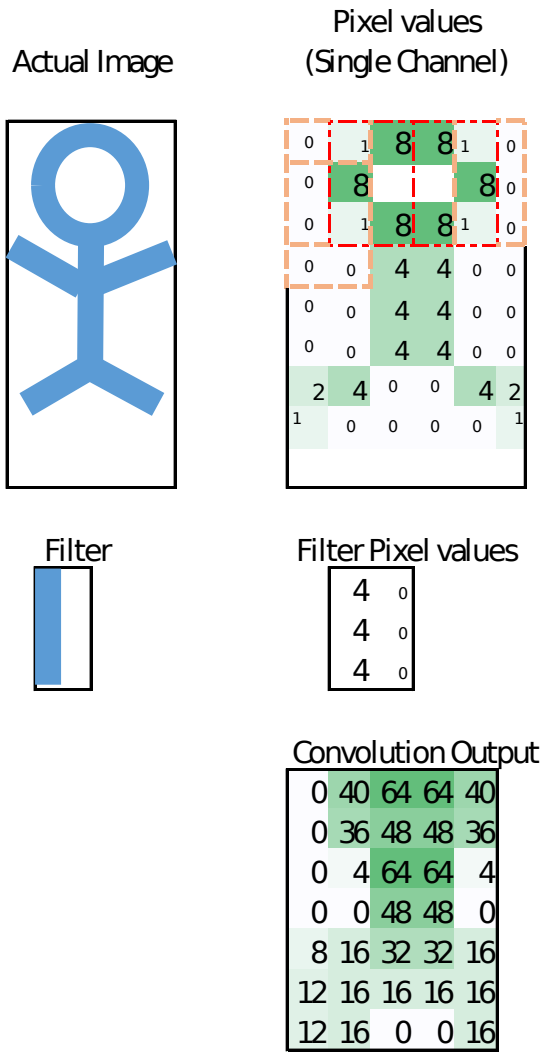
(0*4 + 0*4+0*4)+ (1 * 0 +8*0+1*0) = 0

Op2: Output -40

(1*4 + 8*4 +1*4) + (8*0 + 0*0+**0) = 36

Example:

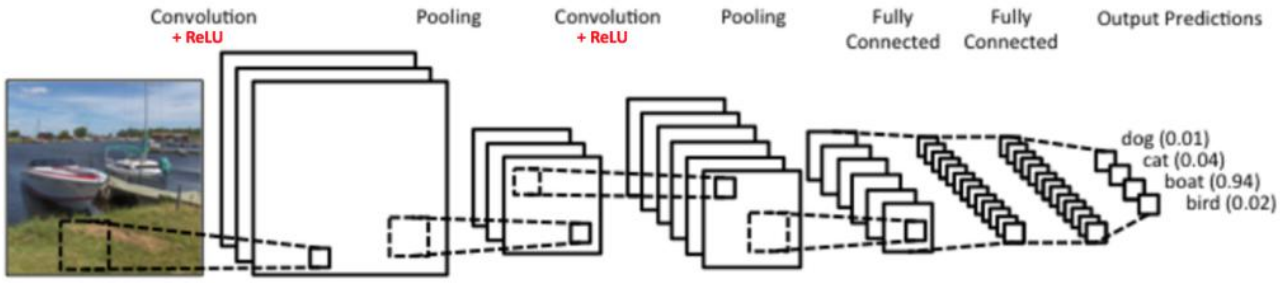
Please note that the below is for demonstration purposes to aid the reader in understanding how convolution works.



The above output is called a feature map. As it can be seen the filter tries to detect straight lines in the image. Since this image is low resolution (the image was essentially a 9x6 – 54 pixels), the output is not very clear. But we can still make out the body of the stick man, which is the straight line has been captured pretty well. In addition the convolution has resulted in a smaller matrix or feature map. The initial 9x6 matrix has been reduced to 7x5 activation map.

I will not go into the details, but convolution has certain parameters which have to be selected when implementing the CovNet – Stride, Padding. However, these are modifications to the above operation which the reader can easily understand.

The output from the convolution layer is fed into a ReLU layer.

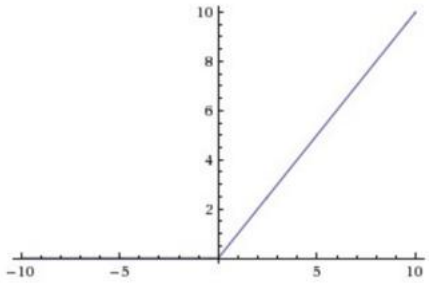


ReLU layer:

A ReLU layer stands for Rectified linear Unit which is a non-linear function. Its function is to introduce non-linearities so the system can generate more complex or richer feature maps.

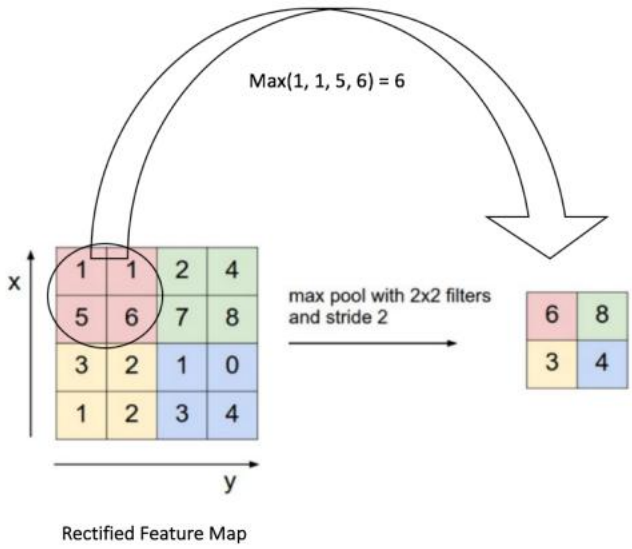
As it can be seen below, a ReLU function retains a positive value but any negative values are forced to zero. This adds the afore mentioned complexity and starts to generate non-linear behavior in the model.

$$\text{Output} = \text{Max}(\text{zero}, \text{Input})$$



Max Pooling Layer:

The pooling layer is designed to reduce the dimensionality of the input feature map while capturing the vital information from the feature map. So essentially it acts as a dimensionality reduction technique for images akin to PCA (Principal component analysis). There are multiple ways to do this – Max, Average Sum etc. I will try to illustrate the Max Pooling layer here as an example.



A rectified feature map is reduced using a max pooling layer to reduce dimensionality of the feature map.

This process of convolution (with multiple filters), ReLU activation and Max Pooling is done multiple times in most of the state of art architectures – resulting in the deep CovNets.

Output layer:

The fully connected layer is a traditional Multi-layer perceptron layer which functions as the classification layer. The MLP layer usually uses a soft max function to compute probabilities of an image belonging to any of the "N" categories.

Benchmark:

The project in this case is similar to the Imagenet challenge in terms of the scale of the project. However, the number of categories is pretty high, more than 5000. In addition, not every category might have enough samples (Images), hence I am not expecting performance matching the Imagenet performance (high 90's). In addition, I have reduced my data set and consequently the number of categories of prediction. The subset dataset will have ~50 categories on which I will train and test. Currently, each category has only 4 images each. Since the input images per Category is only 4 per category- I will randomly choose one image as the test set image for that category. I will use image augmentation increase the sample size to at least 66 images per category resulting in an overall image count of 3300images overall.

Each category, after image augmentation has 66 images. I have separated this into the training and the validation folder in a 10:1 ratio (60 in the training set and 6 in the validation set). Therefore the test set will have a single image for each category which was never seen by the ML algorithm.

Hence a Naive classifier will have a Categorization accuracy of 1/50 or 2% (assuming no learning was involved and its a probability of being anyone of the 50 categories). This is the benchmark I will try to outperform with the final image classifier.

METHODOLOGY:

Data Preprocessing:

The first step is preprocessing the train and the test data sets to get it ready to be fed in the model. Before training the images, the images have to be zero centered. In practice, there are two ways to go about this - subtract the mean image (pixel wise) from all the images to zero center the images in the data set or find the mean for each of the channels (RGB) and subtract that mean from all the images. I intend on using the latter as it is computationally efficient, a limited scalar subtraction instead of a matrix level subtraction for all the images.

For the category labels, I intend to use one hot encoding to code the labels back to the binary label mode.

I intend to use the trained Resnet model and add an additional layer to ensure the correct number of nodes in the output layer. I will be using "Accuracy" as the metric for tuning the multi-class classifier

Implementation:

I have used Keras, a high level neural networks library which has pre-built layers for CovNets to implement the solution.

I had tried two approaches to the problem:

1. Start with a CNN trained from Scratch
2. Use transfer learning to use a sophisticated CNN proven to work on an image dataset and customize a small portion of the CNN to be used as the image recognition tool

Steps taken:

1. Data Extraction from the given BSON files – explained above
2. Data preprocessing including Image augmentation – explained above
3. Divide in to train and test splits after sub setting the problem
    1. Initially I had tried to attempt to classify all available categories – 5200 in number, which proved to be a lot for my laptop to handle or AWS (low cost) machines to handle

2. After consultation with a Udacity mentor I reduced my Categories to 30 which turned out to be a problem which met the resource constraints.
3. To calculate metrics, I ended up splitting the available images into train, validation and test folders.
    a. Since I took only product categories with 4 images, they were split in a 3:1 fashion with 3 images in the training folder and 1 in the testing folder
    b. After the initial split was done, image augmentation was performed to increase sample size and account for shear, rotation, width and height shifts etc. for the images in the training folder. After that, some of the images were moved to the validation folder for hyper parameter tuning. Once the initial model was ready, the model was tested on the unseen data set. The results turned out well, as discussed in the results section.

4. Train a Classifier from scratch and measure Categorization Accuracy
    a. I trained a simple CNN with the following parameters
        i. Layers – 8
            1. Convolution layers – 3
            2. Max Pooling layers – 3
            3. Global Average Pooling – 1
            4. Fully connected output layer – 1 ('Softmax' activation)
        ii. Filters – 3 filters one for each convolution layer
            1. The number of filters increased as one progressed through from the start to the end. Number of filters were 16, 32 and 64 for the layers
            2. Kernel Size – 2
            3. Padding – 'same'
        iii. Activation function used - ReLU
        iv. Metrics – Accuracy
        v. Loss – Categorical Cross Entropy
        vi. Optimizer – Adagrad
5. Use transfer learning from Inception model to measure Categorization accuracy
    a. Model:
        i. I have used InceptionV3 from Google for this Image Classification as my base model for transfer learning as it has good performance for relatively low computational cost
    b. Data set size:
        i. The dataset size for the current challenge is pretty small – each category has a maximum of 4 images each. Therefore I believe it is useful to use a state of the art CNN Model which was trained on a larger dataset and apply it to this current image dataset
    c. Tuning model layers:
        i. The pre-trained model would have learned most of the low level features and high level from the images which it was trained on.
        ii. The idea would be to use the trained and tuned network for the convolution layers and apply it directly to the current problem at hand. Therefore I will freeze the convolutional layers. This has a twofold advantage
            1. We gain from the tuning of the network from a bigger image dataset. The implicit assumption here is that the learning from a big image dataset which is comparable to the current challenge is applicable to the current problem in terms of the features learnt
            2. We save on time and computation of a huge CovNet, wince we will not need to re-train all the layers from scratch
        iii. However, not everything can be leveraged from the State of the art CNN model. The current challenge has 50 output categories, after sub setting the problem whereas the State of Art CNN model might have been used to predict output categories in the range of 1000 – a completely different problem.
        iv. Therefore we need to remove the final classification layers and add a customized layer to reflect the number of categories which we will have. In addition it might be

useful to preface the final classification layer with a fully connected layer and tune its weights. This is necessary, since the output high level features present in the last layer might have to be trained and weighted differently for our current problem. Therefore I have added an additional fully connected layer in addition to the classification layer.

Complications and changes required to the problem statement:

Initially I had started the project as a big data image classification activity involving ~5200 output categories. I soon realized that my laptop was incapable of handling the activity because of memory limitations. I tried moving the challenge to AWS. However, even though it worked better, I still ran into memory issues without increasing the instance class to the high end. So after a lot of experimentation and money spent on AWS, I decided to reduce the scope of the challenge after having a 1:1 appointment with a Udacity mentor.

However, the time spent gave me an understanding of how AWS works – how to run an instance and taught me a lot about running models on the cloud.

Improvements of Algorithms and Techniques:

- I had tried two models for transfer learning - Resnet and Inception V3. On the initial run, the Inception V3 model gave me a better results out of the box - 50%categorization accuracy over the 40% accuracy from the Resnet model. Hence I chose to stick with the InceptionV3 model for further tuning of hyper parameters.

I have focused more on the training of the last few convolution blocks to tune the model for the new image set.

Hyper parameters tuned:

Activation Function - Multiple activation functions can be used - Sigmoid (output ranges between 0 and 1- suitable for the output classification layer), ReLU (most widely used activation layer), Tanh (Like a sigmoid function - but output ranges between -1 and 1). On research, I found that ReLU was the most suitable activation function, used widely in the industry - does not suffer from the vanishing gradient problem seen with sigmoid activation function. Therefore , I ended up using the ReLU activation function for all the intermediate layers as done in most of the CNN architectures.

Optimizer: There are multiple optimizers which can be used. - Adagrad, Nesterov momentum, Adadelta etc. I chose Adagrad as the algorithm automatically decays its learning rate over time. In addition, I had tried Ndam and Nesterov momentum as well. They performed similarly, with Adagrad performing better. In addition Adagrad controls and reduces the learning rate automatically. This was very important for the final tweaking of the top convolution layers, where the modifications to weights had to be small.

Epochs - Epochs indicate the number of times the training data is cycled through for the weights of the optimization network to take effect. In my case, I ended up with 20 epochs. The first epoch gave me a validation accuracy of 94%. After that it was diminishing returns all the way up to the 20^{th} epoch, where I got the validation accuracy to be 98%. I had to stop there because of computation requirements.

I spent the computation time to tune the layers of convolution layers instead of trying different optimizers, since I was limited by the computational complexity. Initially, I had only tuned the top most layer of the new model built on the InceptionV3 model. This resulted in an accuracy of around 50%. However, I then started tuning the model for the last few blocks of convolution layers as well - Layers 249 and up. This resulted in a Categorization accuracy of 69%.

Results:

Model Evaluation and Validation:

I used transfer learning with Inception V3, a CNN Architecture with the second highest image recognition accuracy in the Imagenet challenge, with a relatively modest computational cost[1].

- The test set had an accuracy of 69% much better than the benchmark model
- The model was tested on all categories (one for each of the categories) – 50 in all.
- The model results were robust, when tested with slight variations as well.

Justification:

The model results on the test set of the images is ~70%. This I believe is a pretty good result with the number of images given per category - 4 per category.

- I believe that this is possible only because of using the Inception V3 model which was trained on a huge data set.
- On the flip side, the data category of the Imagenet challenge (on which Inception V3 was trained on) is not overlapping with the product categories. Therefore the metrics might not be as good as the Imagenet challenge results. However when compared to the benchmark model which was trained from scratch, it the results are acceptable.
- I believe that the model is robust - After initial training, the model was then validated on 10 sets of images from the augmented validation set - each containing a random mix of 25 images each.
    - The metrics from the run is reported below. As it can be seen the model seems to be robust on the multiple validation sets - with an average accuracy of 95.2% with a small standard deviation of 3%

Robustness run results:

Test accuracy: 96.00%
Test accuracy: 100.00%
Test accuracy: 96.00%
Test accuracy: 92.00%
Test accuracy: 92.00%
Test accuracy: 100.00%
Test accuracy: 96.00%
Test accuracy: 96.00%
Test accuracy: 92.00%
Test accuracy: 92.00%

Metrics: 95.20% (+/- 2.99%)

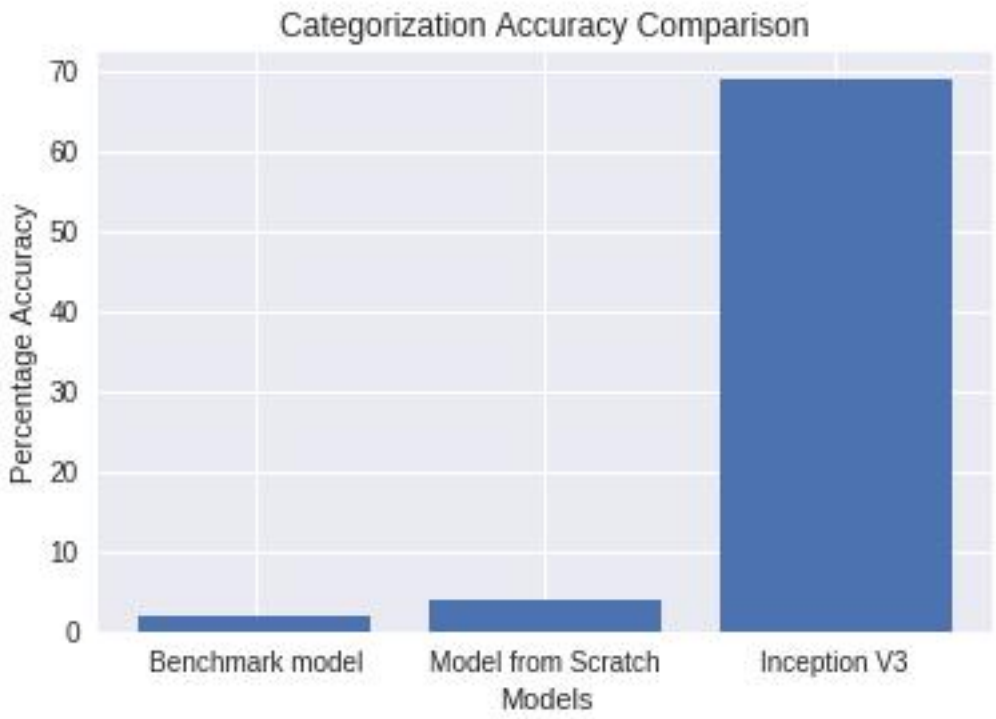
Conclusion:

Free-Form Visualization:

The below chart compares the Categorization Accuracy of the Classifier -

- Benchmark model
- A model trained from scratch

- Model using Transfer learning using the Inception V3 base model



Reflection:

Initially, when I started the project, I intended to tackle the entire data set with about ~5200 output categories as a big data image classification problem. The idea was to then use image augmentation to increase the sample size, for each of the product categories - shear, rotation, zoom and lateral shift. This augmented image data set would then be used to train a CNN for image classification. I started with a benchmark model - a naive assumption of the same output category for all the test set. This would be followed with a simple CNN trained from scratch to illustrate the requirement for a really deep State of the art CNN model, which has good accuracy with a modest computational requirements. Finally I compared the performance of the two models which I believe is respectable.

The biggest learning for me was the scale of the image recognition project I initially took up. I did not anticipate the computational power required to handle such a project. Therefore after discussion with a Udacity coach, I reduced the scope of the problem to 50 output categories. I spent about a month, understanding about AWS and how to run an instance on AWS. This I believe was the biggest payback from this project.

Improvement:

There are two facets of the project which I believe can be improved.

- The number of output categories to be predicted can be increased to the original ~5000 categories
  - Run the classification model on an AWS instance with a high performance GPU
- The output accuracy, I believe can be improved with more training of the earlier layers of the InceptionV3 model. I did not have enough GPU power to train the earlier layers

REFERENCES:

https://www.kaggle.com/c/cdiscount-image-classification-challenge

https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

https://keras.io/preprocessing/image/

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html