

Machine Learning Engineer Nanodegree

Capstone Project

Nikhil Dhiman

March 19th, 2020

DEMO URL : <https://go.aws/2QsJhqi>







I. Definition

Project Overview

In OLX we build leading destinations for buying, selling, and exchanging products and services, spanning across 5 continents and used by 350 million people every month.

Currently, if a customer in OLX wants to post a product, they have to select a category first. Selection of categories is where the major portion of time customers spend. We did some research on the why part and found some interesting pointers raised by customers.

- *I don't understand what you mean by categories*
- *As you have 40 odd categories to select from, it is a hard time to find the relevant category*
- *My definition of categories is not aligning with your definition of categories*

CHOOSE A CATEGORY		
	Properties	>
	Cars	>
	Furniture	>
	Jobs	>
	Electronics & Appliances	>
	Mobiles	>

This is where we as OLX see the opportunity to use ML to solve the customer problem of selecting the product category automatically. Through the image classification algorithm, OLX wants to read the picture that the seller inputs to automatically classify the product into its correct category. Having the correct category for the product will make the experience easier for OLX customers to post.

In this project, I have created a web app capable of accepting images and able to provide the accuracy score for each class. This app uses a convolutional neural network trained on the OLX dataset scrapped by me and present in my submission. The dataset consists of more than 92000 images equally distributed among classes/category.

Problem Statement

The goal is to build a web application that asks the user for product image, and outputs the most likely category that product belongs to, basically providing the accuracy score for each category.

- Taken the set of images with category labels from OLX.
- Train a convolutional neural network that can recognize these images. Expose neural network to a front end application via an AWS Api Gateway and Lambda written in nodejs.
- Build a web application that allows a user to input this application a image of size 112X112px and get an output from the neural network about what category this image belongs to.

Metrics

Accuracy is the metric that is generally chosen to maximize for this CNN.

The training considers validation accuracy maximization at every training iteration as its priority. A metric such as F1 score or log loss is also preferable, but it is important to note that F1 score and log loss are derivable directly from the accuracy score. In such a case, it is preferable to pick accuracy as the defining metric. It should also be noted that the entire dataset contains an almost equal number of data points for each class, and as such does not contain any class imbalance.

Accuracy at every step is determined by the equation:

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{dataset size}}$$

There are 3 accuracy variables to consider here.

- Training accuracy :- This pertains to how accurate the model is when labelling data it has already been trained upon. This accuracy is low in the initial stages of the training period, but gradually converges to 99 percent over the course of the training period

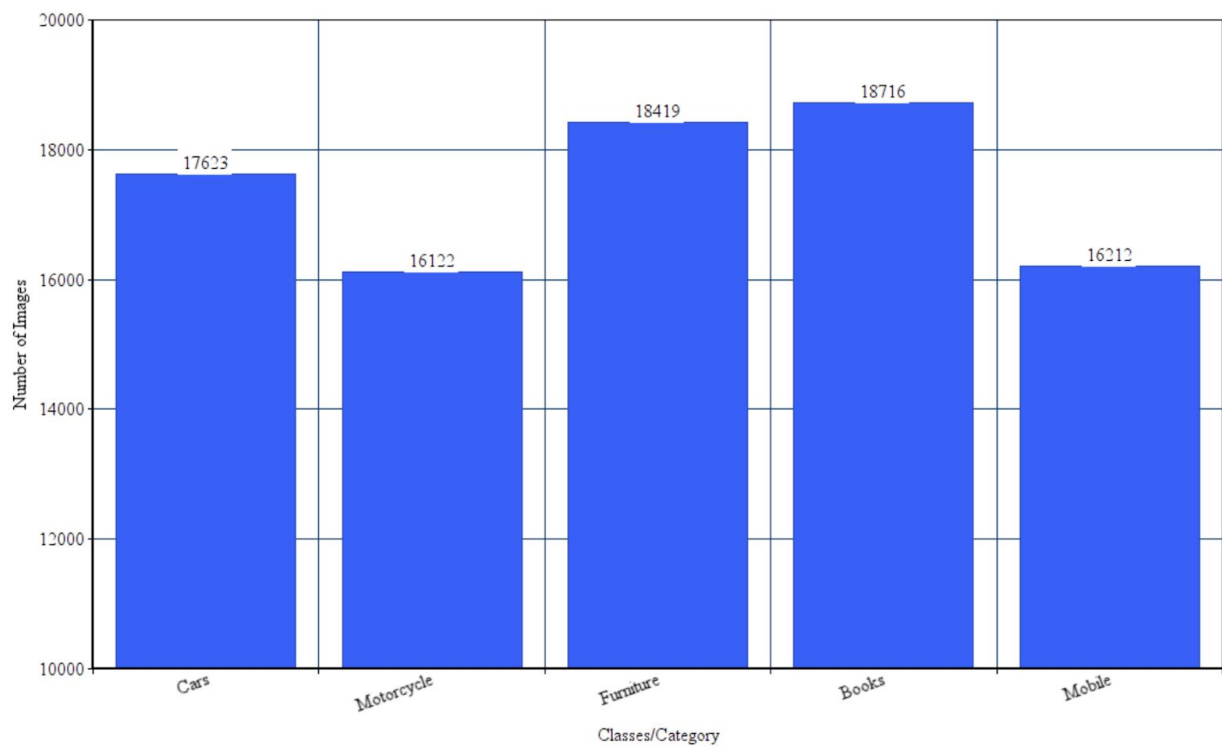
- Validation accuracy :- This metric is the one that the model is actually trained to maximize. Since the validation set is kept isolated from the training set, only those models that show an improvement in validation accuracy are checkpointed, and further training only continues to build on top of the previous model checkpoint.
- Batch accuracy:- Batch size controls the accuracy of the estimate of the error gradient when training neural networks

II. Analysis

Data Exploration

- Every input image provided to the model for training is of resolution 112x112 pixels
- Each image is a RGB
- The entire dataset consists of more than 86K images There are a total of 5 classes that have 86K images which are split equally.
- The Amazon SageMaker built-in Image Classification algorithm requires that the dataset be formatted in RecordIO. RecordIO is an efficient file format that feeds images to the NN as a stream. Since I had the raw images in the filesystem. Then I have to convert the raw images to RecordIO and upload the .rec file to AWS S3.
 - Unzip the data.zip
 - Contains category wise folder and csv that contains links to category specific images
 - Loop over the folder and download the images and convert them to 112x112px RGB format in the same folder to keep them organized.
 - Create two .lst files using a RecordIO tool (im2rec). One file is for the training portion of the dataset (70%). The other is for testing (30%).
 - Generate both .rec files from the .lst
 - Copy both .rec files to an Amazon S3 bucket.

Exploratory Visualization



This plot describes the number of images per unique category in our dataset. As we can see from this plot, there are almost equal distributions of images per category.

Here's a visualization of five sample each from every unique label:

CARS



MOTORCYCLE



BOOKS



FURNITURE



MOBILE



Algorithms and Techniques

This project is a pure classification problem. To solve it, I have chosen to use a convolutional neural network or a CNN. A convolutional Neural network is a state-of-the-art system used to solve data classification problems, along with having other image processing applications. This

approach requires a large amount of data, as compared to other approaches, which seeing the size of the dataset should not be a problem. The training algorithm outputs an assigned probability for each label or class. This is then used to reduce the number of false positives by using a threshold. (The tradeoff here is that this method leads to an increase in the number of false negatives.)

Tuning hyperparameters, depending on how good accuracy can become :



- Number of layers in the Neural network.
- Training length (number of epochs)
- Batch size (how many images to look at, simultaneously during each training step)
- Type of Solver (what algorithm is to be used for learning)
- Learning rate (how fast to learn. This can be dynamic)



I will use a pretrained model that already trained over million of images of different categories, that will give me even higher accuracy.

Benchmark





Primary benchmark will be the accuracy of prediction, I had used Keras deep learning library pre-trained model **ResNet50** and **InceptionResNetV2**. **ResNet50** has shown to achieve an astounding 84% accuracy and InceptionResNetV2 with 60% accuracy. The models and hyper-parameters are set as the same and hardware is also the same.

InceptionResNetV2

Image	Category/Label	Prediction	Time(ms)
	Car	61%	30
	Motorcycle	55%	31

	Furniture	55%	30
	Mobile	59%	27

ResNet50

Image	Category/Label	Prediction	Time(ms)
	Car	85%	33
	Motorcycle	78%	32
	Furniture	68%	32
	Mobile	82%	31

III. Methodology

Data Preprocessing

Performed the following steps to preprocess data:

1. Unzip the data.zip
 - a. This creates folder data and inside it will have 5 folder by the name of categories
 - b. Each category folder has data.csv which contains images url
2. Extract images from URL, convert them to RGB color mode
3. Convert size of images to 112x112px
4. Store them in category folder
5. Now that we have all the images in their respective directories, one per class, it is time to create the RecordIO file. RecordIO is an optimized file format that will feed our images to the Neural Network during training.
6. I will split the dataset into training (70%) and testing (30%). To do that, I will run a python script (im2rec), which is the best tool for this job.

Implementation

I have split my implementation in 2 stages

1. The Training stage
2. Application development stage

Training Stage

1. *Download and Extraction*
 - a. Read data in RGB, normalize and split them into training and testing sets.
2. *Exploration*
 - a. It's all about exploring the dataset and generate visualizations
3. *Data preparation*
 - a. Convert the image to RGB and 112x112px size and then convert the images to RecordIO, and finally upload them to Amazon Simple Storage Service (Amazon S3).
4. *Model training*
 - a. Will select the hyperparameters of my training environment. These parameters will determine how my model will be trained and, consequently, how my trained model will behave
 - b. Define my hyperparameters
 - c. I will submit a job for Sagemaker to train model using my dataset that I uploaded to S3
 - d. Second I will pack the job output into model ready to be used

- e. Tune and repeat step from b to check if accuracy is improved and If I see or attain my required accuracy, I will move to the next step.
 - f. Then I will create an Endpoint Configuration, which is the metadata used by Sagemaker to deploy my model
 - g. Finally I will deploy your model using the Endpoint Configuration
5. *Prediction*
- a. Create an endpoint configuration. In this step, I decide which instance will support my predictions. It is possible to train your model in a machine with GPU and then deploy it on a CPU only machine. This will save the cost of running it on CPU rather than GPU
 - b. I will deploy my model , using the endpoint configuration. SageMaker will create an instance and deploy a container with the algorithm. Inside that container, my trained model will be hosted. After that, I will be able to send requests to that endpoint by using an API.

Application Stage

1. Configure API Gateway
 - a. API gateway to support POST method that accepts base64 encoded data and will proxy to lambda function
2. Configure Lambda Function
 - a. Lambda accepts call from API gateway and call the SageMaker API to predict the category and forward it back to API Gateway
3. WebPage
 - a. WebPage that accepts images from Users, converts that image to 112x112px image and encodes it with base64 and posts it to the endpoint of API Gateway.

Refinement

Phase first: The initial model produced an accuracy of 14.5% and because of that my prediction was never right. While debugging, I saw an error that I made during the data exploration part, in which I store images in a sub directory of category[eg. data/car/images/img_x.jpg], which results in the wrong class set for image.

Phase two: After solving above issue, everything ran smooth and I got accuracy of 49% which was low again, then I started playing with the hyper parameters

- Change number of layers from 18 to 34
- Change epocs from 2 to 4 and then to 6 and after that no improvement in accuracy.
- Enabled the transfer learning

- Changed the learning rate to exponential decay instead of keeping it constant so that it learns fast initially and then slowly over time.

After implementing these, my model now predicts with 99% accuracy

IV. Results

Model Evaluation and Validation

After doing numerous refinement, I was able to achieve

- Validation accuracy :
- Train Accuracy :
- Batch Accuracy:

If the Train accuracy is too different from the Validation-accuracy, probably the model is overfitting, which is not a good thing. A high “Validation-accuracy” indicates that the model is converging to a good format and can generalize well for new images that aren’t part of the training dataset.

According to my benchmark, this model produced much more accurate predictions.

Final model parameters:

- Number of network layers at 34
- Learning rate decay at 0.05
- Used SGD optimizer
- Used accuracy score as our benchmark.
- Output classes at 5

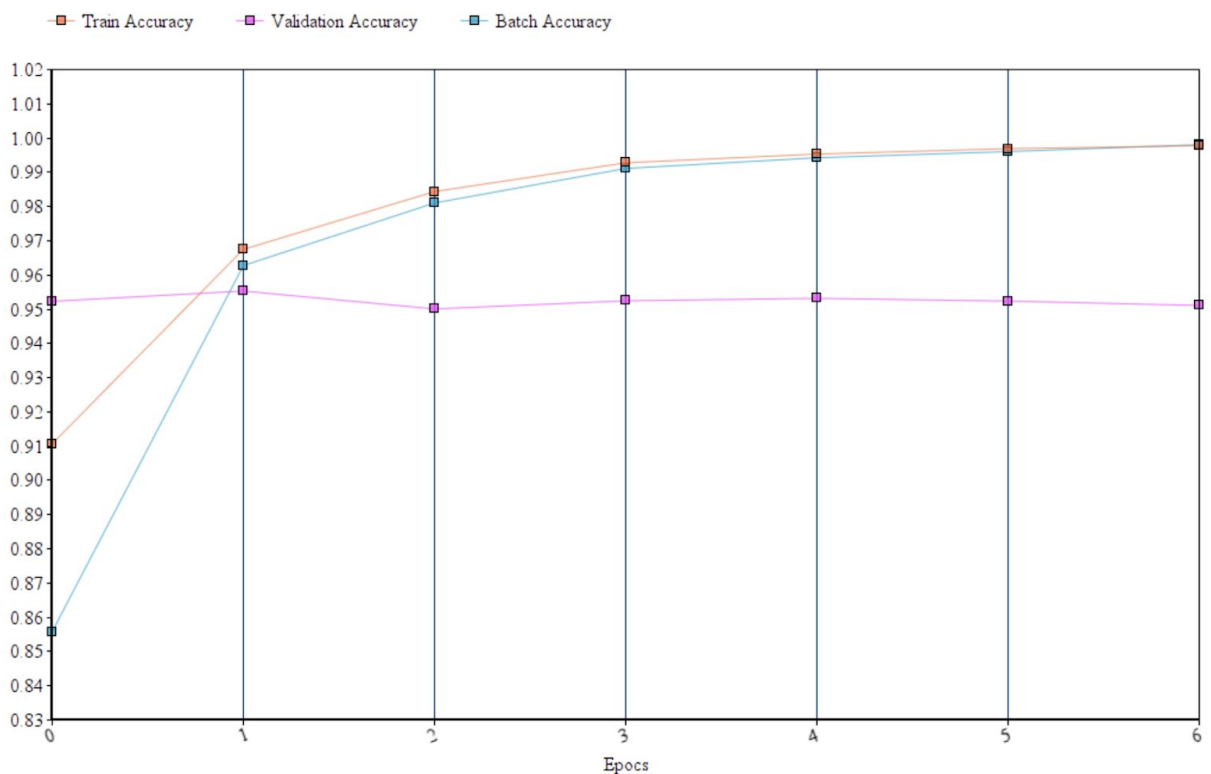
Justification

Benchmark to detect categories of given image is 99% and which is much better than the keras deep learning library[ResNet, InceptionResNetV2]. As my model accuracy is already 99%, I don't feel adding more images in the dataset will help me in any case.

Conclusion

Free-Form Visualization

Here in this following graph, the iterative process of plotting the training, validation and batch accuracy has been represented:

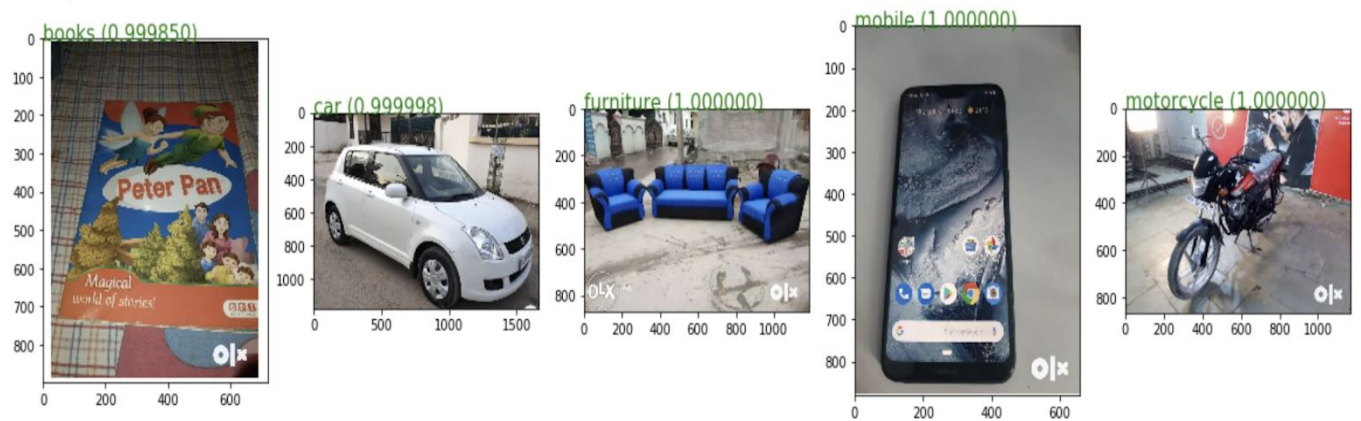


To test how well our model is working, I downloaded random images from OLX and used it for our prediction. Here is the actual images and their labels:



These images were then passed through our model and here is what our model predicted the values to be

```
[0.9998500347137451, 4.873583293374395e-06, 0.000130535670905374, 1.4367052244779188e-05, 1.0494022006923842e-07]
books (0.999850)
[6.676772024150068e-09, 0.999997615814209, 4.4591327963416916e-08, 2.440585376461968e-06, 1.6897381271974155e-08]
car (0.999998)
[4.932523278711187e-10, 1.0844769526840992e-09, 1.0, 3.167728834796435e-08, 3.601324100044323e-11]
furniture (1.000000)
[1.0361862192928561e-12, 1.7355898619328403e-12, 2.100476476779578e-12, 1.0, 8.295634144894759e-13]
mobile (1.000000)
[3.4090340210565957e-15, 1.3726294406657402e-12, 2.2773260707764642e-11, 4.242176475008873e-14, 1.0]
motorcycle (1.000000)
```



Reflection

To summarize, I have successfully built 2 stages of my project.

- First being able to predict the right category for a given product image by using CNN.
- Second part being able to create a basic Web Application using JQuery, Api Gateway, Lambda and S3.

This will help in making the audience understand the capabilities of CNN without having to go through the technical aspect.

Personally for me I had 3 most challenging part of this project:

- How to make sure that Web Application gets response within 1 sec no matter how big the image size is. To optimize network flow, I have to use canvas to resize the image to 112x112px before sending it to Server.
- Data exploration part, faced many challenges like downloading images parallel, im2rec compatibility issues. Image resizing and changing to RGB mode. Disc size full.
- Turing the hyperparameters and knowing exactly when I am overfitting or underfitting. While training the model. Choosing an appropriate machine for training, most of the time I got the segmentation fault error and got it solved by going 1 level up on the machine.

Improvement

There are several ways I can improve efficiency and effectiveness.

- Dataset size: Having the same accuracy even if image size is reduced to 28x28 from 112x112
- Training time: As I played with hyper parameters, a lot of time was wasted in training the model. Using a better GPU like NVIDIA Tesla V100 by changing the instance type from P2 to P3 will reduce the training time significantly. V100 is 14 times faster than K80
- Training time: To further improve training time over huge dataset, I can use Sagemaker instance_count functionality to distribute the load among machines.

I am sure that there is nothing called the best solution, if we discover the way to train models on low end machines with reasonable time, greater possibilities will emerge.