

### **Ecommerce Product Category Identification**

**Given a product image determine with x% confidence whether the product belongs to a particular category (mobile for this exercise).**

### Acknowledgements

I would like to convey my heartfelt thanks to Mr Neeraj Bisht, Senior Technologist at a leading E-commerce company for the problem statement and guiding me throughout this project.

## **Project: Image Classification in E-commerce**

<b><i>Classification : What is it?</i></b> .....	<b>3</b>
<b>Image Classification?</b> .....	<b>3</b>
<b><i>Neural Networks : Deep Learning</i></b> .....	<b>3</b>
<b><i>Convolution Neural Networks(CNN)</i></b> .....	<b>4</b>
<b>Residual Learning(resnet18)</b> .....	<b>5</b>
<b><i>Data Collection : How?</i></b> .....	<b>6</b>
<b>Python</b> .....	<b>6</b>
<b>Pytorch, Resnet18</b> .....	<b>7</b>
<b><i>Procedure in Brief</i></b> .....	<b>7</b>
<b><i>Experimental Data</i></b> .....	<b>8</b>
<b><i>Analysis</i></b> .....	<b>10</b>
<b><i>Visualizations</i></b> .....	<b>11</b>
<b>Insights and Explanations</b> .....	<b>13</b>
<b><i>Conclusion</i></b> .....	<b>14</b>
<b><i>Bibliography</i></b> .....	<b>15</b>
<b><i>Appendix</i></b> .....	<b>16</b>
<b>Appendix A : Trial 8 Data Table</b> .....	<b>16</b>
<b>Appendix B : Trial 2-7 Data Table</b> .....	<b>20</b>
<b>Appendix C : Python based Algorithm for Image Classification</b> .....	<b>29</b>
<b>Appendix D : Personal Reflection</b> .....	<b>35</b>

### **Classification : What is it?**

Classification is one of the most widely used methods of identifying trends in datasets. These types of algorithms work towards the objective of classifying datasets reliably without employing other data manipulation techniques. Classification algorithms are either Supervised or Unsupervised. Supervised learning uses labelled data to make decisions whereas the latter uses unlabeled data. For E.g., Supervised learning may contend with linear relationships whereas unsupervised will have to contend with unknown relationships (usually clustering). Therefore, depending on the type of dataset, the analyst will have to decide on the appropriate algorithm to employ for a problem. For this study, supervised learning algorithms have been chosen as the dataset is labelled.

### **Image Classification?**

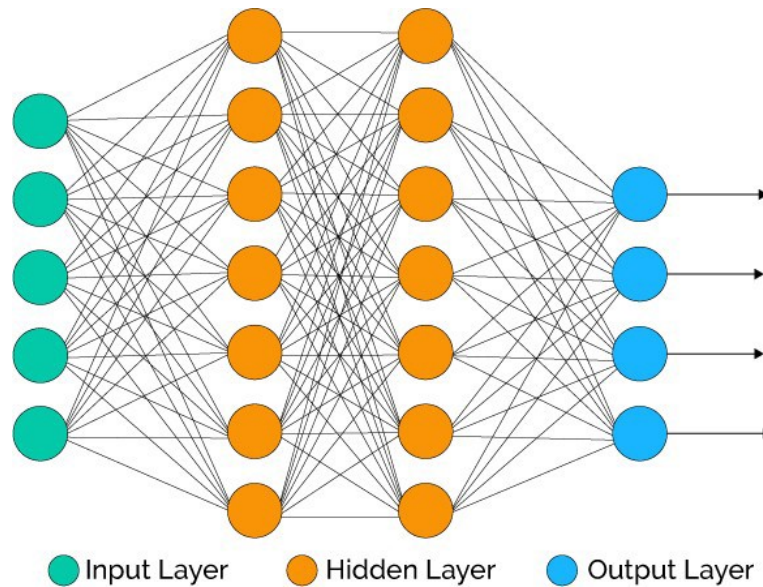
Image Classification is the machine learning process by which a user is able to categorize a set of images into different classes or themes. Using a model (trained or untrained), images can be provided so that the model uses neural networks with deep learning techniques to classify all images accurately. The model is a pre-coded set of instructions for the computer to apply to a set of images so that it is able to classify the dataset into classes/categories.

### **Neural Networks : Deep Learning**

Neural networks are an attempt to simulate brain function. The brain contains trillions of neuron connections whereas, developers are able to program only a limited number of connections between nodes in the artificial neural network. The artificial neural networks, consisting of an input layer, output layer, activation layer and several other compute/hidden layers, is a large decision tree which uses mathematical methods to compute millions of

results in order for the algorithm to progress in the training, validation or even the testing phase.

Below is a simple illustration(annotated) of a neural network:



(Fig 1: Neural Networks(NN) Diagram)

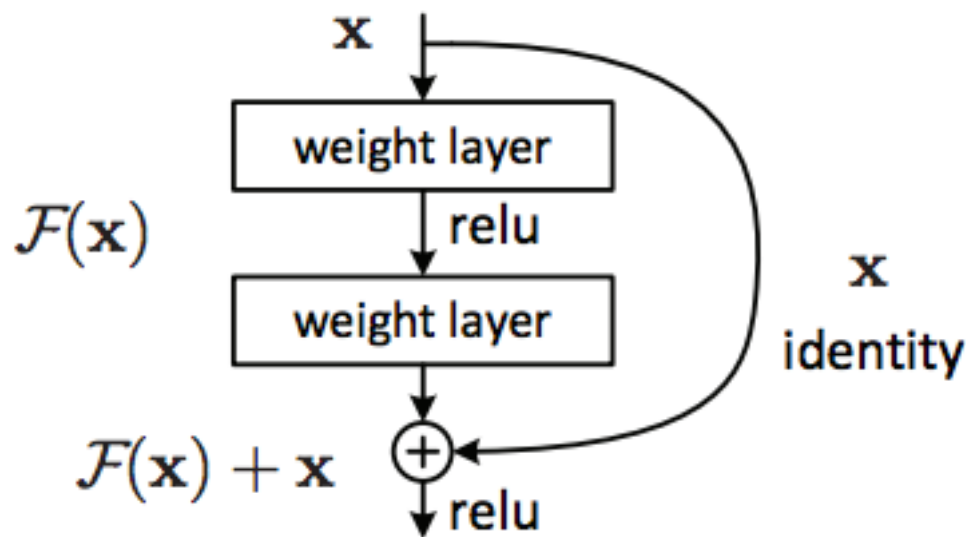
### **Convolution Neural Networks(CNN)**

“Convolution” refers to the process of using 2 functions to create a new third one.

“Convolution Neural Networks” is a neural network which uses two given inputs and produces an output as it moves to a node in successive hidden layers, the activation layer or the output layer. Also, the CNN is used in the context of residual learning. Residual learning is the process by which the neural network is able to re-use any inputs it had lost in a successive layer which makes the algorithm produce the least loss and a higher accuracy.

### Residual Learning(resnet18)

Below is a diagram portraying the process by which CNN is able to train a certain dataset:



(Fig 2: Single Process in the CNN)

In the above diagram,  $F(x)$  is the initial function and the  $x$  is the loss which is carried forward in order to reduce loss and improve accuracy. The weight layer refers to a step in the process where the mathematical calculation is carried out in order to progress to the next hidden layer.

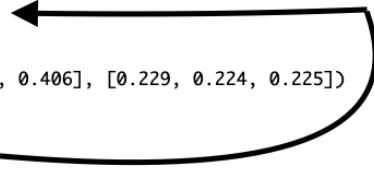
## Data Collection : How?

This exploration has used *Python* with *Pytorch* and *resnet18*.

### *Python*

*Python* is a high-level programming language, which uses simple syntax to perform difficult mathematical operations making it useful for complex deep learning processes involving large neural networks(with numerous computational layers between I/O layers). Below is an annotated excerpt of Python code for this study.

```
data_transforms = {  
    'train': transforms.Compose([  
        transforms.RandomResizedCrop(224),  
        transforms.RandomHorizontalFlip(),  
        transforms.ToTensor(),  
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  
    ]),  
    'val': transforms.Compose([  
        transforms.Resize(256),  
        transforms.CenterCrop(224),  
        transforms.ToTensor(),  
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  
    ]),  
}
```



Resizing Every Image  
in the Training  
and Validation dataset

(Fig 3: Using code to prepare data for training and cross-validation phases)

As seen in Fig1, using a few lines of Python, many images can be prepared for training which otherwise would have taken more time and effort manually. Python has a large reservoir of neural networks and deep learning libraries which are crucial tools in image classification. Pytorch is developed by Facebook AI Research Labs(FAIR) is compatible with Python.

## **Pytorch, Resnet18**

Pytorch is open-source and contains pre-trained and un-trained learning models which make it suitable for most ML developers. *Resnet18* is the machine learning model for deep learning algorithms. “18” refers to the 18 layers of the CNN. This project has utilized the pretrained version Resnet18 so that the process can be simplified to the real-time training and validation of the given images.

### **Procedure in Brief**

1. Ensure adequate computing power and memory
2. Install Python and PyTorch using command line
3. Run a few commands to verify that Python and PyTorch are functioning as intended
4. If, your computer does not have adequate computing power, follow steps 5 to 7
5. Contact an expert in Industry or Setup a temporary desktop
6. Configure the desktop to be a desktop server
7. Use either “ssh” or a cabled connection to run the Python commands on the Desktop
8. Record values for training and validation accuracy, time
9. Iterate algorithms by changing values of learning rate and Epoch# until optimal training and validation accuracy is achieved.

### **Experimental Data<sup>1</sup>**

This project attempts to classify a sample of 500 mobile and non-mobile images minimizing the error rate.

<b>Notation</b>	<b>Description</b>
lr(Learning Rate)	Determinant for the rate at which the loss function converges
Step Size	Similar to lr in the sense it determines how quick a model gets used to the dataset
Momentum	Directs features towards specific elements of the dataset to ensure quick convergence.
Epoch#	Refers to the number of iterations for classifying the dataset of Images
Time(s)	Refers to the time taken for the algorithm to completely classify the given dataset of Images

<b><i>Trial 1</i></b>						
lr	1	Epoch#	Train Loss	Accuracy	Val Loss	Accuracy
Step Size	7	0	12.8921	0.5654	49.6867	0.4299
Gamma	0.1	1	1.1616	0.5473	32.2009	0.4091
Momentum	0.9	2	1.177	0.5626	56.9414	0.5862
Epoch#	5	3	1.0158	0.564	8.5094	0.6234
Time(s)	563	4	1.0218	0.5479	31.9983	0.406

The table above provides a glimpse of the type of data collection done to make informed choices about the values of learning rate(lr) and Epoch#.

---

<sup>1</sup> Data for Trials 2-7 are given in the Appendix B



<i>Table of Constants</i>				
Step Size	7	7	7	7
Gamma	0.1	0.1	0.1	0.1
Momentum	0.9	0.9	0.9	0.9

Below is a condensed set of data tables which consist of key statistical measures for each trial and combination of lr and Epoch# used. For example, has 5 iterations and took 563 seconds with an overall training accuracy of 56%.

Trial#	1	2	3	4
lr	1	0.1	0.01	0.01
Epoch#	5	10	15	20
Time(s)	563	1142	1713	2284
Median <sup>2</sup> Train Loss	1.1616	0.6759	0.6524	0.3464
Median Train Acc	0.5626	0.6211	0.6454	0.86075
Median Val Loss	32.2009	24.30385	15.9971	0.25525
Median Val Acc	0.4299	0.6022	0.6168	0.90475

---

<sup>2</sup> Val: Cross Validation, Acc: Accuracy.

Trial#	5	6	7	8
lr	0.01	0.01	0.01	0.01
Epoch#	25	50	75	90
Time(s)	3454	5713	8576	10324
Median Train Loss	0.2447	0.1617	0.2825	0.0853
Median Train Acc	0.9067	0.9378	0.8917	0.9781
Median Val Loss	0.158	0.3156	0.2142	0.00675
Median Val Acc	0.9356	0.9422	0.927	1

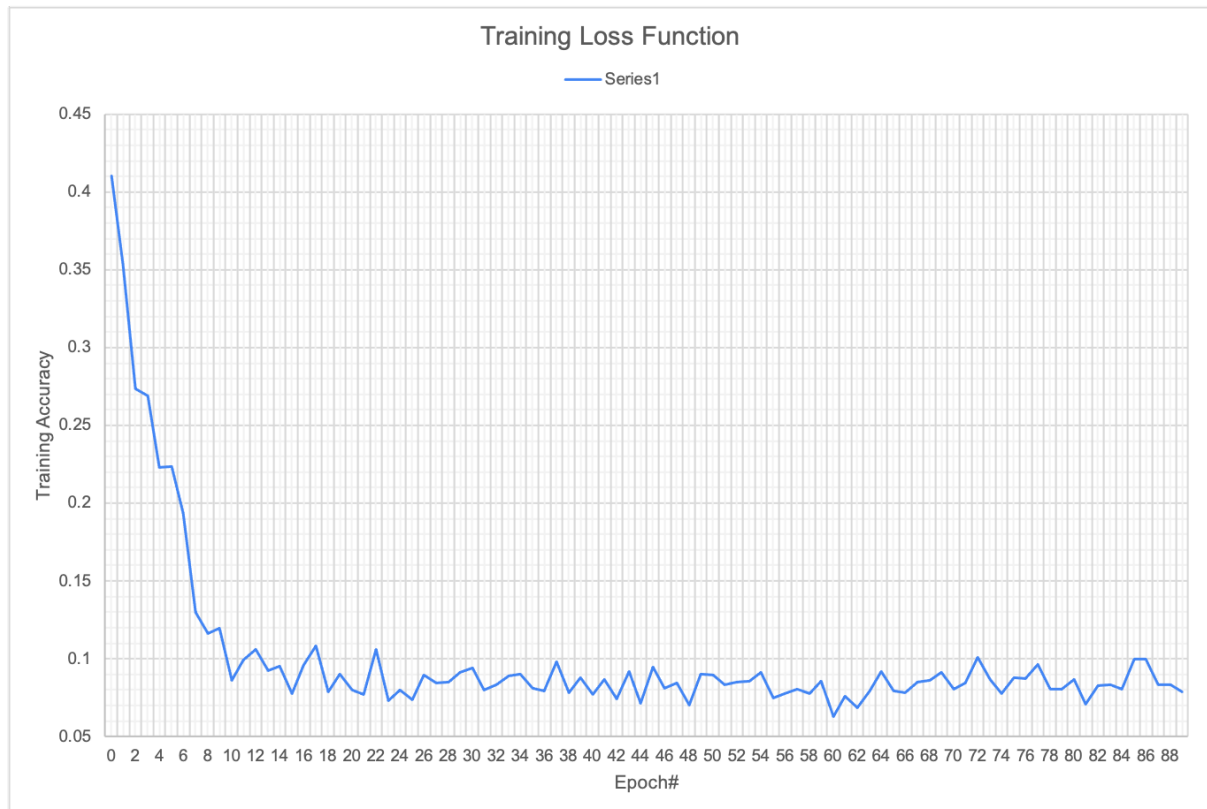
### **Analysis**

Trial 8 has produced the least loss and highest overall accuracies on both training and cross validation datasets. The combination of Trial 8 may produce the same measurements on other datasets. Trial 8 has been further detailed as that combination enables the algorithm to perform the best in comparison to the others.

<b><i>Key Statistics for Trial 8</i></b>	
Median Train Loss	0.0853
Median Train Acc	0.9781
Median Val Loss	0.00675
Median Val Acc	1
Mean Val Acc	99.8216854

## Visualizations

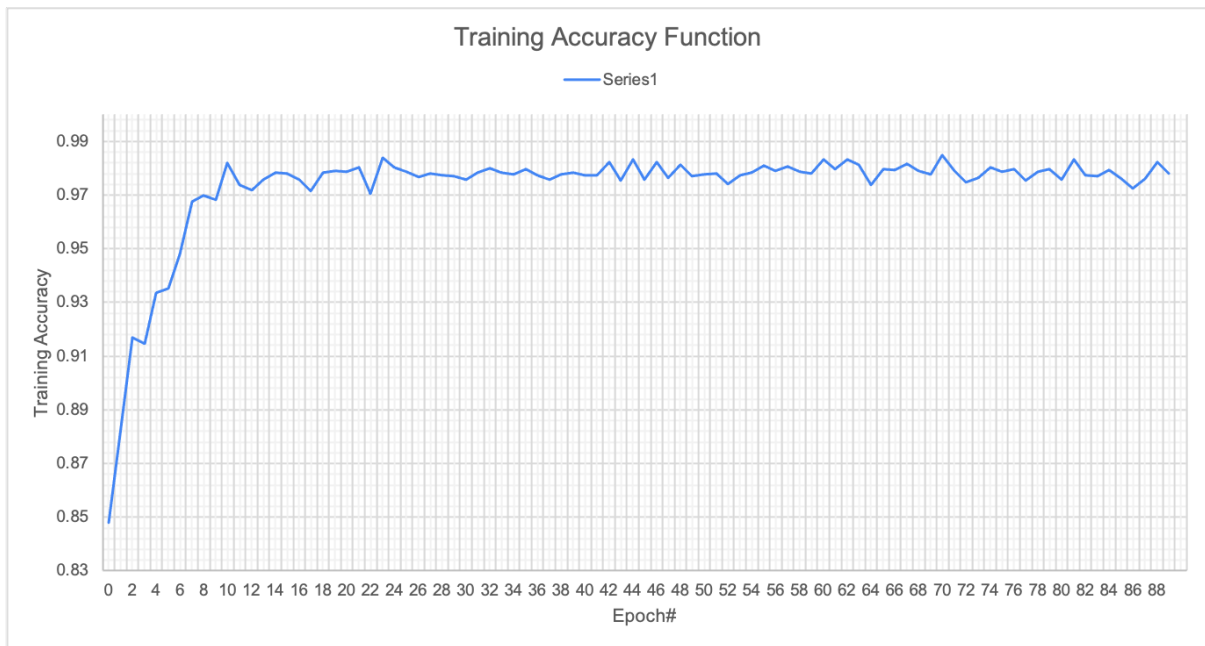
Shown below are the visualizations related to Trial 8<sup>3</sup>.



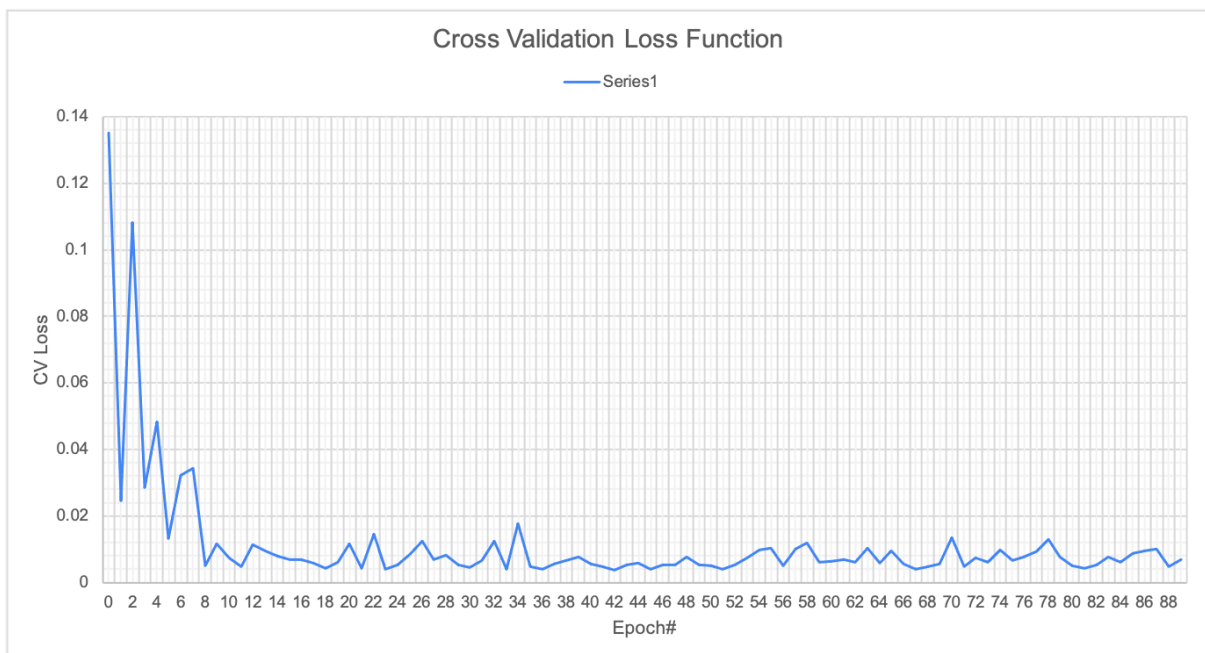
(Fig 4 : Training Loss Function for Trial 8)

---

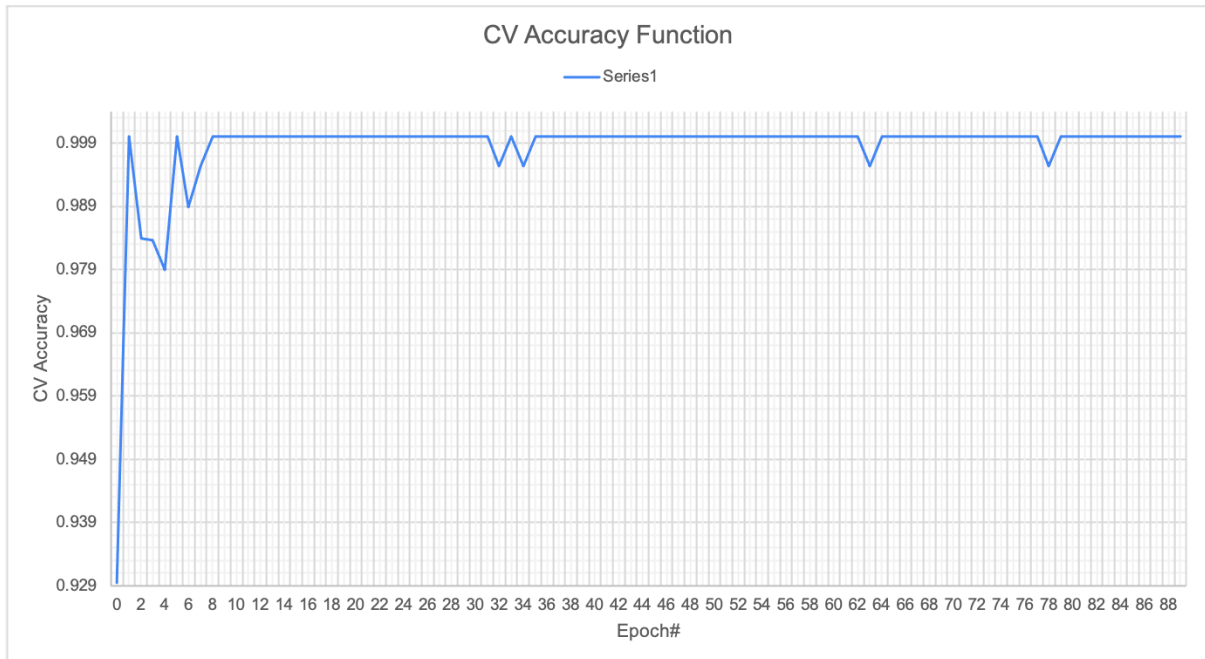
<sup>3</sup> Fully detailed data for Trial 8 is given in Appendix A



(Fig 5 : Training Accuracy Function for Trial 8)



(Fig 6 : Cross Validation Loss Function for Trial 8)



(Fig 7 : Cross Validation Accuracy Function for Trial 8)

### Insights and Explanations

Fig#	Description/Explanation/Insights
4,5	<p>Pertains to the training phase of the model on the dataset. As seen in fig 4, the training loss sharply decreases in the first section of the graph. This might indicate a “warm up” phase for the model so as to understand the feature vector and comparison basis. This could be a similar case for the training accuracy, however, in the case of training accuracy the accuracy values spike quickly which indicates that the model did not use a large number of iterations to achieve the higher accuracy values. Although, one aspect of the graph which is a limitation of using the choices as is, is the perceivable asymptote at 0.99 for the accuracy and the amount of fluctuations in the accuracy values over the 90 iterations. This is an aspect to consider while making choices for learning rate and epoch# as a minor</p>

	<p>difference in the values can produce a significant different in accuracy and the time taken for the model to completely classify the dataset. This can be seen by comparing the absolute values for the measurements of Trial 7 and Trial 8 in the table above containing median values for all measurements.</p>
6,7	<p>The visualizations for cross validation portray a similar trend to that of the training phase. However, in the cross validation the performance of the model is marginally better. This shows that the training phase has been effective because the performance on the training set generally provides the developer with crucial information about the performance of the same on the cross validation dataset. In this scenario, the cross validation is seen as the test set which meant that receiving 100% accuracy in majority of the iterations indicates success for the model as it shows it is able to solve the problem with high precision.</p>

### **Conclusion**

To summarize the problem statement was “Given a product image determine with x% confidence whether the product belongs to a particular category (mobiles for this exercise)”. The algorithm classifies images correctly with an high, overall accuracy of 99.82168%. This indicates that the combination for trial 8 has been successful. However, there is still an error rate of almost 0.28%, when applied on larger datasets could mean a significant absolute error. Hence, improvements and extensions to this project are:

1. Manipulating step size and momentum of the CNN.
2. Using a different model. E.g., TensorFlow.
3. Applying the images in different orientations to test unpredictable data. E.g., rotation
4. Applying a larger dataset to the model to improve data collection

## **Bibliography**

- Bhatia, Richa. “Neural Networks Do Not Work Like Human Brains – Let's Debunk The Myth.” *Analytics India Magazine*, 31 Oct. 2018, [analyticsindiamag.com/neural-networks-not-work-like-human-brains-lets-debunk-myth/#:~:text=b\)%20Artificial%20neural%20networks%20vs,there%20are%20millions%20of%20connections.](https://analyticsindiamag.com/neural-networks-not-work-like-human-brains-lets-debunk-myth/#:~:text=b)%20Artificial%20neural%20networks%20vs,there%20are%20millions%20of%20connections.)
- Bonner, Anne. “The Complete Beginner's Guide to Deep Learning: Convolutional Neural Networks.” *Medium*, Towards Data Science, 1 June 2019, [towardsdatascience.com/wtf-is-image-classification-8e78a8235acb](https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb).
- Facebook AI Research Lab(FAIR). “Torchvision.models¶.” *Torchvision.models - PyTorch 1.7.0 Documentation*, Torch Contributors, 2019, [pytorch.org/docs/stable/torchvision/models.html](https://pytorch.org/docs/stable/torchvision/models.html).
- Fung, Vincent. “An Overview of ResNet and Its Variants.” *Medium*, Towards Data Science, 17 July 2017, [towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035](https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035).
- Jain, Prachi. “Image Classification for E-Commerce [Part I].” *Medium*, Towards Data Science, 13 Feb. 2019, [towardsdatascience.com/product-image-classification-with-deep-learning-part-i-5bc4e8dccf41](https://towardsdatascience.com/product-image-classification-with-deep-learning-part-i-5bc4e8dccf41).
- McDonald, Conor. “Neural Networks(NN) Diagram.” *Towards Data Science*, Medium, 22 Dec. 2017, [towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef](https://towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef).
- Python.org. “Welcome to Python.org.” *Python.org*, Python.org, [www.python.org/doc/](http://www.python.org/doc/).

Ruiz, Pablo. “Understanding and Visualizing ResNets.” *Medium*, Towards Data Science, 23 Apr. 2019, [towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8](https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8).

Zhou, Victor. “Machine Learning for Beginners: An Introduction to Neural Networks.” *Medium*, Towards Data Science, 20 Dec. 2019, [towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9](https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9).

## **Appendix**

### **Appendix A : Trial 8 Data Table**

	Learning Rate = 0.01		Epoch# = 90	
Epoch#	Train Loss	Train Accuracy	Val Loss	Val Accuracy
0	0.4104	0.8481	0.1352	0.9295
1	0.3532	0.8811	0.0245	1
2	0.2734	0.917	0.1081	0.9839
3	0.2688	0.9147	0.0284	0.9836
4	0.223	0.9336	0.0483	0.9789
5	0.2238	0.9353	0.0132	1
6	0.1933	0.9481	0.0323	0.9889
7	0.13	0.9675	0.0344	0.9953
8	0.1162	0.97	0.005	1
9	0.1194	0.9681	0.0117	1
10	0.0861	0.9819	0.0073	1
11	0.0994	0.9739	0.0048	1
12	0.1061	0.9717	0.0113	1



13	0.0924	0.9758	0.0095	1
14	0.0954	0.9783	0.0079	1
15	0.0779	0.9781	0.0069	1
16	0.0956	0.9758	0.0068	1
17	0.1082	0.9714	0.0058	1
18	0.0785	0.9783	0.0042	1
19	0.0899	0.9789	0.0061	1
20	0.08	0.9786	0.0116	1
21	0.077	0.9803	0.0043	1
22	0.1059	0.9706	0.0144	1
23	0.0732	0.9839	0.0039	1
24	0.0797	0.9803	0.0053	1
25	0.0739	0.9786	0.0084	1
26	0.0895	0.9767	0.0123	1
27	0.0847	0.9781	0.0069	1
28	0.0849	0.9772	0.0083	1
29	0.0914	0.977	0.0053	1
30	0.0942	0.9756	0.0044	1
31	0.08	0.9783	0.0066	1
32	0.0834	0.98	0.0125	0.9953
33	0.0888	0.9783	0.0041	1
34	0.0903	0.9778	0.0178	0.9953
35	0.0812	0.9797	0.0048	1
36	0.0791	0.9775	0.0041	1

37	0.0982	0.9756	0.0056	1
38	0.078	0.9778	0.0067	1
39	0.0881	0.9783	0.0077	1
40	0.0771	0.9775	0.0056	1
41	0.0867	0.9772	0.0048	1
42	0.0741	0.9822	0.0038	1
43	0.0919	0.9753	0.0053	1
44	0.0715	0.9833	0.0057	1
45	0.0948	0.9756	0.0039	1
46	0.0809	0.9822	0.0052	1
47	0.0843	0.9764	0.0054	1
48	0.07	0.9814	0.0076	1
49	0.0902	0.977	0.0053	1
50	0.0897	0.9778	0.0049	1
51	0.0832	0.9781	0.004	1
52	0.085	0.9742	0.0053	1
53	0.0858	0.9775	0.0073	1
54	0.0914	0.9783	0.0098	1
55	0.0747	0.9811	0.0103	1
56	0.0779	0.9789	0.0049	1
57	0.0806	0.9806	0.0101	1
58	0.0776	0.9786	0.0118	1
59	0.0856	0.9781	0.0062	1
60	0.0627	0.9833	0.0064	1

61	0.0757	0.9797	0.0068	1
62	0.0687	0.9833	0.0061	1
63	0.0791	0.9814	0.0103	0.9953
64	0.092	0.9739	0.0058	1
65	0.0793	0.9795	0.0095	1
66	0.0783	0.9792	0.0055	1
67	0.0848	0.9817	0.0041	1
68	0.086	0.9789	0.0047	1
69	0.0915	0.9778	0.0055	1
70	0.0803	0.985	0.0134	1
71	0.0844	0.9789	0.0048	1
72	0.1011	0.9747	0.0075	1
73	0.0865	0.9764	0.006	1
74	0.0775	0.9803	0.0098	1
75	0.0876	0.9786	0.0067	1
76	0.0875	0.9795	0.0076	1
77	0.0965	0.9753	0.0092	1
78	0.0803	0.9786	0.0129	0.9953
79	0.0806	0.9795	0.0077	1
80	0.0867	0.9756	0.0049	1
81	0.0708	0.9831	0.0043	1
82	0.083	0.9775	0.0052	1
83	0.0832	0.977	0.0078	1
84	0.0804	0.9792	0.006	1

85	0.0996	0.9761	0.0087	1
86	0.0996	0.9725	0.0095	1
87	0.0831	0.9761	0.01	1
88	0.0831	0.9822	0.0048	1
89	0.079	0.9781	0.0069	1

### **Appendix B : Trial 2-7 Data Table**

#### **Trial 2**

Epoch#	Train Loss	Train Accuracy	Val Loss	Val Accuracy
0	0.6799	0.6171	61.672	0.5959
1	0.6821	0.6112	27.2153	0.4138
2	0.6746	0.6143	13.131	0.6407
3	0.6846	0.6071	73.1523	0.5926
4	0.6801	0.609	14.7739	0.6407
5	0.6772	0.6251	62.5074	0.5993
6	0.6739	0.6254	33.7717	0.4132
7	0.6553	0.6418	21.3924	0.6265
8	0.6536	0.6454	5.488	0.6157
9	0.6527	0.6454	15.8623	0.6051

#### **Trial 3**

Epoch#	Train Loss	Train Accuracy	Val Loss	Val Accuracy
0	0.6539	0.6454	7.3349	0.6165
1	0.6532	0.6454	22.9799	0.624
2	0.6536	0.6454	10.9086	0.6184

3	0.6524	0.6454	15.9971	0.6218
4	0.6527	0.6454	6.618	0.6112
5	0.654	0.6454	6.6196	0.6207
6	0.6539	0.6454	40.3398	0.5926
7	0.651	0.6454	18.7793	0.6168
8	0.6574	0.6451	46.7435	0.5882
9	0.6507	0.6454	42.3574	0.5793
10	0.6506	0.6454	11.8881	0.6284
11	0.6508	0.6454	24.6538	0.5937
12	0.6506	0.6454	16.2664	0.6265
13	0.6507	0.6454	5.0082	0.6118
14	0.6508	0.6454	9.5473	0.6234

#### **Trial 4**

Epoch#	Train Loss	Train Accuracy	Val Loss	Val Accuracy
0	1.7928	0.5873	0.9531	0.6332
1	0.7705	0.6307	0.8157	0.6723
2	0.6817	0.6745	0.623	0.7431
3	0.561	0.7312	0.6375	0.757
4	0.5067	0.767	0.5441	0.8176
5	0.457	0.7984	0.3388	0.8725
6	0.4177	0.825	0.2623	0.9067
7	0.3586	0.8489	0.2435	0.9134
8	0.3543	0.8589	0.2724	0.8811
9	0.3539	0.8609	0.2467	0.92

10	0.3389	0.8606	0.3068	0.8959
11	0.3253	0.865	0.2399	0.9178
12	0.334	0.8623	0.2205	0.9253
13	0.3302	0.8636	0.2107	0.9281
14	0.3152	0.8703	0.2482	0.9178
15	0.3074	0.8773	0.2395	0.9028
16	0.3211	0.8773	0.3094	0.8656
17	0.3263	0.8723	0.2158	0.9239
18	0.3081	0.8834	0.2367	0.9181
19	0.3095	0.887	0.2124	0.9292

### **Trial 5**

Epoch#	Train Loss	Train Accuracy	Val Loss	Val Accuracy
0	0.3866	0.8384	0.273	0.8914
1	0.3784	0.8492	0.2486	0.9034
2	0.3475	0.857	0.2365	0.9267
3	0.3368	0.8595	0.6177	0.8625
4	0.3319	0.8689	0.2711	0.8867
5	0.3135	0.8753	0.3623	0.8961
6	0.3133	0.8786	0.3469	0.9131
7	0.2688	0.8917	0.1729	0.9353
8	0.2504	0.9017	0.1734	0.9311
9	0.253	0.8995	0.1422	0.9417
10	0.2509	0.8989	0.1495	0.9375
11	0.2468	0.9067	0.137	0.9481

12	0.2447	0.905	0.1421	0.9397
13	0.2391	0.9092	0.15	0.9356
14	0.2307	0.9095	0.158	0.9422
15	0.2262	0.9103	0.1453	0.9464
16	0.2116	0.9192	0.1281	0.9506
17	0.2176	0.912	0.214	0.9111
18	0.2283	0.9081	0.1183	0.9595
19	0.2206	0.9136	0.1285	0.9483
20	0.2339	0.9139	0.1218	0.9661
21	0.2194	0.9192	0.1742	0.9331
22	0.2164	0.9142	0.1221	0.9528
23	0.2215	0.9131	0.1255	0.9572
24	0.2235	0.9117	0.1779	0.9356

### **Trial 6**

Epoch#	Train Loss	Train Accuracy	Val Loss	Train Accuracy
0	0.2964	0.8842	0.1896	0.927
1	0.2972	0.8809	0.2135	0.9195
2	0.2787	0.8864	0.1654	0.9414
3	0.2561	0.8989	0.2189	0.9306
4	0.2698	0.892	0.2592	0.8928
5	0.2479	0.8984	0.3752	0.9122
6	0.2331	0.907	0.2573	0.9153
7	0.2087	0.9203	0.2706	0.9581
8	0.1976	0.9261	0.2541	0.9342

9	0.1794	0.9314	0.2937	0.9372
10	0.1797	0.9306	0.3338	0.922
11	0.1834	0.9284	0.2786	0.9436
13	0.175	0.9295	0.2032	0.945
14	0.176	0.9322	0.2782	0.9425
15	0.1626	0.9431	0.2503	0.9514
16	0.1677	0.9331	0.3045	0.9592
17	0.1566	0.942	0.2937	0.9608
18	0.161	0.9381	0.288	0.947
19	0.1613	0.9397	0.2937	0.9561
20	0.1532	0.9439	0.3417	0.9447
21	0.1603	0.9389	0.4085	0.9561
22	0.1563	0.942	0.406	0.9264
23	0.1619	0.9359	0.384	0.9364
24	0.1629	0.9386	0.394	0.9264
25	0.1616	0.9406	0.4854	0.9464
26	0.1593	0.9381	0.3369	0.9481
27	0.1662	0.9395	0.3305	0.9525
28	0.1588	0.937	0.3868	0.9353
29	0.1734	0.9292	0.3354	0.9442
30	0.1612	0.9425	0.3032	0.9595
31	0.1539	0.9461	0.3734	0.9431
32	0.1718	0.9359	0.3973	0.9117
33	0.1617	0.9386	0.2626	0.9567



34	0.1612	0.9406	0.3781	0.9384
35	0.1545	0.9403	0.3779	0.9242
36	0.1567	0.9411	0.3551	0.9347
37	0.1634	0.9375	0.3407	0.9514
38	0.1595	0.9406	0.4444	0.9331
39	0.1601	0.9359	0.2731	0.9656
40	0.1506	0.9467	0.5118	0.9286
41	0.1541	0.9356	0.3156	0.9631
42	0.1541	0.9453	0.4435	0.8959
43	0.1566	0.9378	0.2078	0.9631
44	0.1689	0.9403	0.2919	0.9486
45	0.1476	0.9489	0.2833	0.9375
46	0.1612	0.9378	0.328	0.9378
47	0.1583	0.9417	0.3444	0.9472
48	0.1542	0.9428	0.3076	0.9422
49	0.1628	0.9356	0.3266	0.9384

### **Trial 7**

Epoch#	Train Loss	Train Accuracy	Val Loss	Val Accuracy
0	1.4768	0.5948	0.8667	0.6829
1	0.7149	0.6393	0.5913	0.699
2	0.5959	0.6968	0.4873	0.7626
3	0.5295	0.7412	0.699	0.8173
4	0.4585	0.792	0.3994	0.8434
5	0.4447	0.7998	0.4709	0.82

6	0.4081	0.8256	0.2999	0.8898
7	0.3628	0.8545	0.2607	0.9167
8	0.3404	0.8636	0.2529	0.9167
9	0.3329	0.8698	0.2245	0.9297
10	0.3176	0.8723	0.2078	0.9328
11	0.31	0.8798	0.2117	0.9253
12	0.3076	0.8825	0.2111	0.942
13	0.3001	0.8878	0.219	0.9128
14	0.2968	0.8928	0.2376	0.9128
15	0.2906	0.8839	0.2022	0.9325
16	0.2887	0.8914	0.2017	0.9342
17	0.2938	0.887	0.2106	0.9186
18	0.2772	0.8959	0.2164	0.9209
19	0.2812	0.8945	0.2221	0.9131
20	0.2957	0.8853	0.2038	0.932
21	0.2811	0.8886	0.2189	0.9314
22	0.2807	0.8923	0.2118	0.9245
23	0.2908	0.887	0.271	0.915
24	0.2786	0.8903	0.219	0.9314
25	0.2825	0.892	0.2275	0.9147
26	0.286	0.8881	0.2281	0.9197
27	0.2855	0.8942	0.2135	0.9314
28	0.2713	0.8978	0.2142	0.9342
29	0.2838	0.8945	0.2034	0.9439

30	0.2771	0.8986	0.2378	0.9197
31	0.2755	0.8975	0.252	0.9228
32	0.2902	0.8914	0.2082	0.9386
33	0.283	0.8939	0.2003	0.9339
34	0.2714	0.8998	0.2426	0.9317
35	0.2783	0.8906	0.2133	0.9253
36	0.2788	0.8995	0.1992	0.9322
37	0.272	0.8945	0.3144	0.8931
38	0.2731	0.8989	0.2196	0.927
39	0.2784	0.8984	0.209	0.9297
40	0.278	0.8931	0.2041	0.9389
41	0.2842	0.8948	0.2215	0.9295
42	0.2788	0.8911	0.2132	0.927
43	0.2725	0.8959	0.2306	0.9197
44	0.2727	0.8973	0.2035	0.9228
45	0.2819	0.8917	0.2071	0.9342
46	0.2853	0.8914	0.1852	0.9328
47	0.2836	0.8911	0.2198	0.9314
48	0.2699	0.8967	0.2155	0.9314
49	0.2844	0.892	0.2011	0.9317
50	0.2843	0.8923	0.1877	0.9411
51	0.2806	0.897	0.2058	0.9408
52	0.2796	0.8948	0.2007	0.9317
53	0.2704	0.8956	0.2017	0.9247

54	0.2751	0.8881	0.2233	0.922
55	0.2821	0.8939	0.2023	0.9222
56	0.2833	0.887	0.2048	0.932
57	0.2924	0.887	0.2571	0.9289
58	0.2797	0.8909	0.226	0.9247
59	0.2799	0.8928	0.2209	0.9267
60	0.2753	0.8973	0.211	0.9297
61	0.2782	0.8909	0.234	0.9275
62	0.264	0.9025	0.2684	0.9239
63	0.2851	0.8906	0.2507	0.9131
64	0.2675	0.902	0.201	0.9414
65	0.2894	0.887	0.2956	0.8995
66	0.2736	0.8959	0.2064	0.9339
67	0.2758	0.8961	0.2038	0.9342
68	0.2732	0.8961	0.2072	0.9314
69	0.2787	0.8948	0.1989	0.9386
70	0.2847	0.8917	0.2231	0.9246
71	0.2838	0.8911	0.1921	0.9306
72	0.2879	0.8878	0.2076	0.9153
73	0.2816	0.8931	0.1931	0.9239
74	0.2803	0.892	0.2042	0.9306

## Appendix C : Python based Algorithm for Image Classification<sup>4</sup>

```
from __future__ import print_function, division

import torch

import torch.nn as nn

import torch.optim as optim

from torch.optim import lr_scheduler

import numpy as np

import torchvision

from torchvision import datasets, models, transforms

import matplotlib.pyplot as plt

import time

import os

import copy

plt.ion() # interactive mode

print("Testing.....Done")

data_transforms = {

    'train': transforms.Compose([

        transforms.RandomResizedCrop(224),

        transforms.RandomHorizontalFlip(),

        transforms.ToTensor(),

        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

    ]),

    'val': transforms.Compose([

        transforms.Resize(256),
```

---

<sup>4</sup> This Code Has been Adapted from <https://towardsdatascience.com/how-to-train-an-image-classifier-in-pytorch-and-use-it-to-perform-basic-inference-on-single-images-99465a1e9bf5>

```

        transforms.CenterCrop(224),

        transforms.ToTensor(),

        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

    ]),
}

print("Testing.....Done")

data_dir = 'ecommerce'

image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),

                data_transforms[x])

                for x in ['train', 'val']}

dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,

                shuffle=True, num_workers=4)

                for x in ['train', 'val']}

dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}

class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

print("Testing.....Done")

def imshow(inp, title=None):

    """Imshow for Tensor."""

    inp = inp.numpy().transpose((1, 2, 0))

    print("Testing.....Done")

    mean = np.array([0.485, 0.456, 0.406])

    print("Testing.....Done")

    std = np.array([0.229, 0.224, 0.225])

    print("Testing.....Done")

```

```

inp = std * inp + mean

inp = np.clip(inp, 0, 1)

print("Testing.....Done")

plt.imshow(inp)

if title is not None:

    plt.title(title)

plt.pause(0.001) # pause a bit so that plots are updated

# Get a batch of training data

inputs, classes = next(iter(dataloaders['train']))

print("Testing.....Done")

# Make a grid from batch

out = torchvision.utils.make_grid(inputs)

imshow(out, title=[class_names[x] for x in classes])

print("Testing.....Done")

def train_model(model, criterion, optimizer, scheduler, num_epochs=25):

    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())

    best_acc = 0.0

    for epoch in range(num_epochs):

        print('Epoch {}/{}'.format(epoch, num_epochs - 1))

        print('-' * 10)

        for phase in ['train', 'val']:

            if phase == 'train':

                model.train()

            else:

```

```

    model.eval()

running_loss = 0.0

running_corrects = 0

for inputs, labels in dataloaders[phase]:

    inputs = inputs.to(device)

    labels = labels.to(device)

    optimizer.zero_grad()

    with torch.set_grad_enabled(phase == 'train'):

        outputs = model(inputs)

        _, preds = torch.max(outputs, 1)

        loss = criterion(outputs, labels)

        if phase == 'train':

            loss.backward()

            optimizer.step()

    running_loss += loss.item() * inputs.size(0)

    running_corrects += torch.sum(preds == labels.data)

if phase == 'train':

    scheduler.step()

epoch_loss = running_loss / dataset_sizes[phase]

epoch_acc = running_corrects.double() / dataset_sizes[phase]

print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_acc))

if phase == 'val' and epoch_acc > best_acc:

    best_acc = epoch_acc

    best_model_wts = copy.deepcopy(model.state_dict())

print()

```



```

time_elapsed = time.time() - since

print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed %
60))

print('Best val Acc: {:.4f}'.format(best_acc))

model.load_state_dict(best_model_wts)

return model

print("Testing.....Done")

def visualize_model(model, num_images=6):

    was_training = model.training

    model.eval()

    images_so_far = 0

    fig = plt.figure()

    with torch.no_grad():

        for i, (inputs, labels) in enumerate(data loaders['val']):

            inputs = inputs.to(device)

            labels = labels.to(device)

            outputs = model(inputs)

            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):

                images_so_far += 1

                ax = plt.subplot(num_images//2, 2, images_so_far)

                ax.axis('off')

                ax.set_title('predicted: {}'.format(class_names[preds[j]]))

                imshow(inputs.cpu().data[j])

            if images_so_far == num_images:

```

```

        model.train(mode=was_training)

    return

    model.train(mode=was_training)

print("Testing.....Done")

# print out folder name and class name

model_ft = models.resnet18(pretrained=True)

num_fts = model_ft.fc.in_features

# Here the size of each output sample is set to 2.

# Alternatively, it can be generalized to nn.Linear(num_fts, len(class_names)).

model_ft.fc = nn.Linear(num_fts, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

print("Testing.....Done")

# Observe that all parameters are being optimized

optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)

model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=90)

```

### **Appendix D : Personal Reflection**

This has been a great learning experience. I am fascinated to see the way a small list of commands is able to make predictions in a short period of time. Initially, I had difficulty understanding the concepts. However, the guidance from Mr Neeraj Bisht, Machine Learning course on Coursera and articles from *Medium* were immensely helpful. I would recommend a system which contains a dedicated Nvidia GPU based on my experience