Spring 5-20-2019

Optimizing E-Commerce Product Classification Using Transfer Learning

Rashmeet Kaur Khanuja
San Jose State University

Optimizing E-Commerce Product Classification Using Transfer Learning

A Project Report

Presented to

Dr. Robert Chun

Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Class

CS 298

By

Rashmeet Kaur Khanuja

May 2019

The Designated Project Committee Approves the Project Titled

Optimizing E-Commerce Product Classification Using Transfer Learning

By

Rashmeet Kaur Khanuja

Approved for the Department of Computer Science

San Jose State University

May 2019

Dr. Robert Chun	Department of Computer Science
Dr. Nada Attar	Department of Computer Science
Mr. Praveen Thareshappa	Senior Software Engineer, Walmart Labs

ABSTRACT

The global e-commerce market is snowballing at a rate of 23% per year. In 2017, retail e-commerce users were 1.66 billion and sales worldwide amounted to 2.3 trillion US dollars, and e-retail revenues are projected to grow to 4.88 trillion USD in 2021. With the immense popularity that e-commerce has gained over past few years comes the responsibility to deliver relevant results to provide rich user experience. In order to do this, it is essential that the products on the ecommerce website be organized correctly into their respective categories. Misclassification of products leads to irrelevant results for users which not just reflects badly on the website, it could also lead to lost customers. With ecommerce sites nowadays providing their portal as a platform for third party merchants to sell their products as well, maintaining a consistency in product categorization becomes difficult. Therefore, automating this process could be of great utilization. This task of automation done on the basis of text could lead to discrepancies since the website itself, its various merchants, and users, all could use different terminologies for a product and its category. Thus, using images becomes a plausible solution for this problem. Dealing with images can best be done using deep learning in the form of convolutional neural networks. This is a computationally expensive task, and in order to keep the accuracy of a traditional convolutional neural network while reducing the hours it takes for the model to train, this project aims at using a technique called transfer learning. Transfer learning refers to sharing the knowledge gained from one task for another where new model does not need to be trained from scratch in order to reduce the time it takes for training. This project aims at using various product images belonging to five categories from an ecommerce platform and developing an algorithm that can accurately classify products in their respective categories while taking as less time as possible. The goal is to first test the performance of transfer learning against traditional convolutional networks. Then the next step is to apply transfer learning to the downloaded dataset and assess its performance on the accuracy and time taken to classify test data that the model has never seen before.

Index Terms **– convolutional neural networks, deep learning, dropout, e-commerce product categorization, keras, transfer learning**

ACKNOWLEDGEMENTS

I stretch out my appreciation to my advisor Dr. Robert Chun and my board of committee Dr. Attar and Mr. Thareshappa for their constant support throughout the course of my Master of Science Project dissertation and for helping me whenever I needed guidance. Their experience on various subjects assisted me in learning numerous industry practices while actualizing my decisions. Without their guidance and encouragement, it would not have been a conceivable task to complete this venture.

Contents

TABLE OF FIGURES:

LIST OF TABLES

1. INTRODUCTION

1.1. Importance and Aim of the project

The digital world has completely changed the way we shop today. We are facing today what is known as the “retail apocalypse” [1]. It refers to the fact that for two decades now, shopping trends are shifting more and more towards online stores because of the ease of shopping it provides to users. With online shopping, users can easily compare prices from various stores and shop at any time of the day. While this shift is a big opportunity for the e-commerce market, it also adds responsibility to deliver accurate results when users search for something. The accuracy and variety of accurate results that a website shows determines its popularity amongst the users for the kind of experience that they get. Today, ecommerce websites are more popularly known as “marketplaces” where not only customers can buy products, but also register as sellers to sell their items [2]. Though this opens up new opportunities for both the website and users in terms that they get a plethora of different products to sell and buy, it also adds a management overhead on the website to make sure that all vendors or merchants add their products in the same format. If this standardization is not maintained, it leads to inaccurate results. When users investigate a particular category, they expect to find a variety of relevant products within the bracket they’re looking. When searching for a cotton in the beauty/self-care category, finding a kayak in results is a disappointment (Figure 1.). The reasons behind this wrongful product appearance could be many but the consequence of multiple such inaccurate results lead to a search-graveyard, i.e. the customer feels so letdown, they just move to another website to find their choice of item. Therefore, it is very important for customers to be able to find the right products in the category they search for when surfing an ecommerce website.

In a large-scale usability testing experiment conducted by Baymard Researchers [3], it was found that the rate of success of finding products and purchasing them was influenced up to 400% by simple design features such as correct products linked to their respective categories. Also, manual annotation of 110 e-commerce website design samples was done, and its results showed that the ease or difficulty for the users to surf through a website’s catalog is governed by the way products are classified into categories. It is further mentioned that categorization of products is directly proportional to user experience, i.e., more effective the links between products and their respective categories, better is the navigation to the correct page. Numbers like this are evidence enough to state the importance of correct

categorization of products so that when users go to a particular category, they only find related products; and if they search for a particular product, it links back to the right category so that relevant similar results can be shown. Thus, it is apparent that one factor of categorization affects many other features in terms of ecommerce purchase.

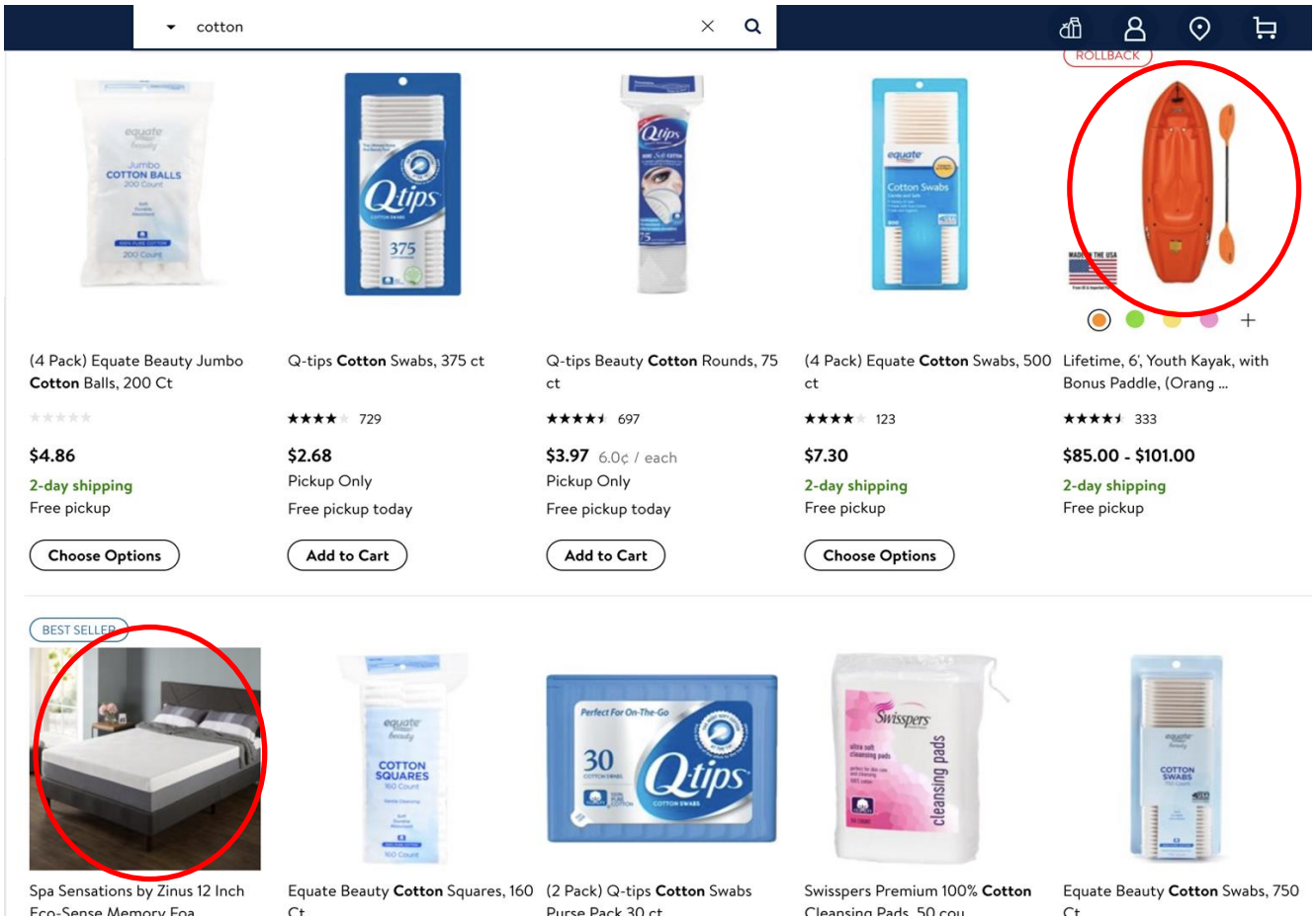

Figure 1. Inaccurate results in cotton category

The current systems in place that help with the automatic categorization of products mainly follow text-based classification approaches [4] [5]. That means whatever title and description text is entered by the sellers, that is used to place the products into their respective categories. Many times, it happens that merchants do not select proper categories owing to too many options to fill up, or do not enter the exact keywords, or even enter extra keywords to boost their item's appearance in user search [5]. This leads to products appearing out of context in the incorrect categories. Also, the textual description of a product on the seller's end and its interpretation on the buyer's end could be completely different. Thus, relying solely on title or description text is not enough, product image needs to be taken into

account for better classification. While some approaches do use computer vision to classify products, they face many challenges such as high training time, low accuracy, requirement of powerful computing resources, etc. [6].

This project aims at using a technique called transfer learning to classify products to appropriate categories based on their images. Transfer learning is a procedure that makes use of previously gained knowledge to apply it further to other related use-cases [7]. For example, a model trained to identify dogs can be used to identify other animals or images. The benefit of using this approach is the time, cost, and effort it saves. When there are millions of images available, the dataset becomes so large that training a model from scratch takes hours and also requires very powerful machines including GPUs where images are involved. On the other hand, when such an extensive dataset is not available, conventional neural networks do not train well for the lack of enough samples. Transfer learning helps to solve both of these contingencies. Since it uses pre-trained models such as VGG-16, VGG-19, ResNet, etc., they do not need to be trained from scratch, only their last layers need to be fine-tuned. This process is a lot faster than training a Conventional Neural Network (CNN) from scratch. Furthermore, when enough data is not available, which is likely in case of ecommerce websites where we would not always find millions of images for each category, transfer learning provides data augmentation capabilities that help increase the amount of available data and make the model learn better. A renowned professor and data scientist Andrew Ng mentioned in his tutorial 'Nuts and bolts of building AI applications using Deep Learning' saying, "After supervised learning — Transfer Learning will be the next driver of ML commercial success".

This project also aims to compare time taken and accuracy attained to train a model using traditional CNN and VGG-16. This survey uses references from published papers, statistics from surveys, conference proceedings, and some real-life events.

2. BACKGROUND

Ecommerce market's explosive growth has created an overload of information on the users that visit the site. With the growth of online shopping, and addition of third-party sellers on selling platforms, maintaining a standard taxonomy of products is becoming more and more important to provide the users the ease to find products of their choice.

This growth of ecommerce has been exponential in the past few years. Users that surf these online websites to fulfill their shopping needs come from all demographics. Be it a 60-year-old or a 6-year-old, today's technological advancements have made it favorable for all age groups to come to a search engine, look for desired items in a category or specify what they want in text or speech, and get a list of results to choose from. The accuracy and relevance of this result list determines how happy users will be.

Ecommerce giants such as Amazon, Walmart, eBay, etc. list millions of products on their websites. To present these products to users in an organized manner so as to simplify search and navigation, they are categorized into multi-level categories [5]. For example, Electronics -> iPads and Tablets -> iPads -> Apple iPad 32 GB. This way the end product Apple iPad 32 GB falls under multiple levels of hierarchy. This sort of hierarchical taxonomy makes the pathway to reach to the desired items easier and users then know exactly where to navigate to find what they need. Ecommerce websites today do not only display their own products but also items from merchants who register as sellers on their websites. Merchants need to enter all the product information manually and not all merchants take the time and effort to do so. Also, various merchants have their own taxonomies the way they store their product information, these individual taxonomies need to be unified into a single canonical taxonomy for the website [8] (Figure 2.). This kind of standardization is necessary to provide users a seamless searching and shopping experience.

While some products on ecommerce websites are categorized manually, this number is very small [6]. Having millions of products on the website, it is not possible to manually categorize each and every one of them, which is why an automatic classification system becomes of utmost importance for any ecommerce website. There are different approaches that can be adopted to automate this classification. Most of the existing approaches use text in order to classify products to their respective categories [9] [10]. Classification done solely on the basis of text suffers from problems such as missing/incomplete text, text unrelated to product, text related to multiple categories, and

inconsistent vocabulary used by different merchants. This leads to misleading information for the automation model that then classifies products in inaccurate categories. An example of this is two perfectly valid textual descriptions “Asus Laptop *with* Battery” and “Asus Laptop Battery” for a laptop and a battery respectively. Based on the given description, a text-based model would classify them both the same category even though one belongs to laptops and the other to accessories.

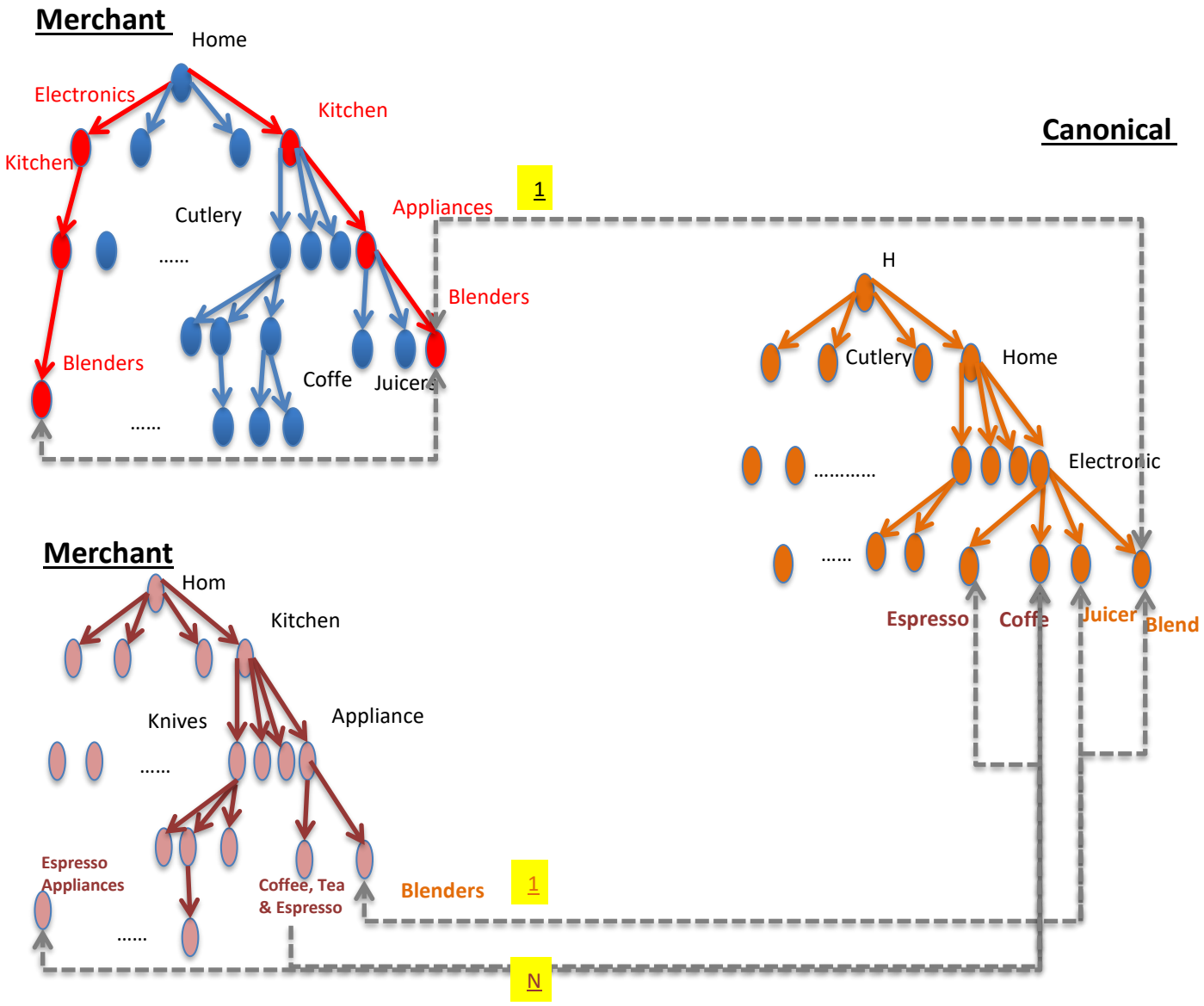

Figure 2. Different merchant taxonomies that need to be unified into one
Source: Adapted from [8]

In order to perform more relevant categorization, images can be used instead of or in combination with text. The idea behind using images is that in most cases, users shop only after looking at the product image, and even though the textual description of a product could be incorrect, a user would only buy

something that they see to be correct. Thus, classification done on the basis of images can lead to correct categorization of products int their respective categories. This would not only help users with easy navigation but also save a lot of their time and enhance experience. As stated by [11], if we can reduce the time spent by users on searching for results by a mere 1% by providing them with relevant results in the first place, we will save 187,000 person-hours each month.

To make this happen, we propose the use of an image-based machine learning approach for product classification. Machine learning techniques like computer vision have been used in the past for image classification. Training any machine learning model anew on a huge dataset is a very time taking process and requires robust machines. Therefore, we propose a technique that uses a machine learning model that has already been trained on a huge dataset and gained relevant knowledge that can be used on our current dataset with some modifications. This approach, known as transfer learning, helps save time and provides equally good accuracy.

Figure 3 shows the organization of the literature survey. It begins with a brief explanation of the ecommerce product classification methodologies out there. The paper further delves into optimizing product classification using neural networks. The next section focuses on how this classification can be made faster and accurate using the proposed approach of transfer learning using image data to categorize products and a comparison of the proposed approach with the conventional methods.

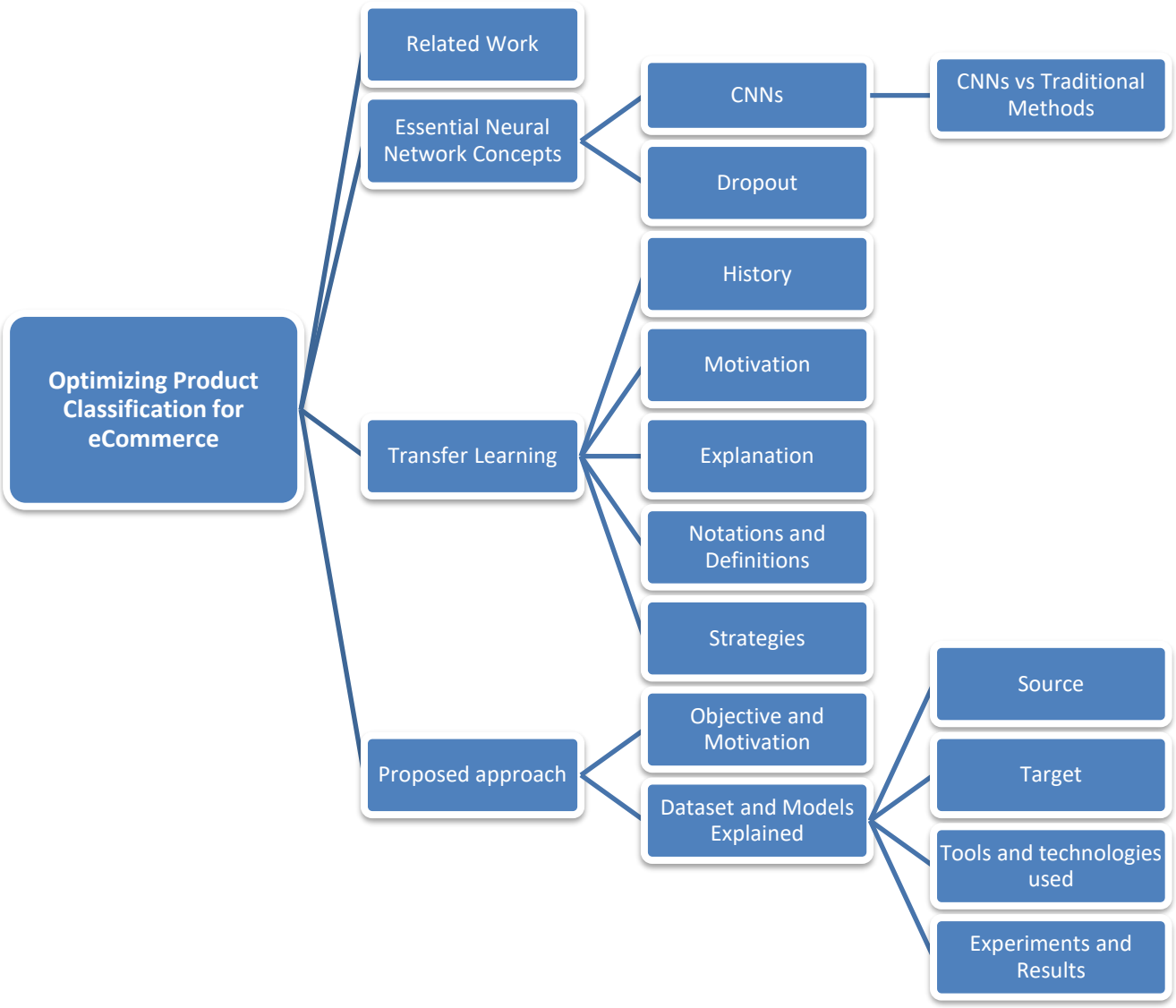

Figure 3. Organization of the Literature Survey

3. RELATED WORK

This section explains what the existing methodologies are and what endeavors have been taken by others towards ecommerce product classification. Though there are several approaches that have been adopted towards automating the categorization of ecommerce products, there is not any documentation to show the use of the particular technique of transfer learning in the ecommerce domain.

R. Fishkin and M. Staff [12] explained the early phase of ecommerce product classification when it was based completely on keyword matching. This was a very early automation technique to assign products to different categories based on keywords in its title or description. This method is not very accurate since merchants do not actively fill up all information details of a product or sometimes they add too much and too irrelevant information just to boost their products being displayed to the users. Another approach proposed by Z. Kozareva [13] is to take into account not just the product title but also the taxonomy along with it to discern the corresponding category. R. Agrawal and R. Shrikant [9] spoke about integrating documents and categorizing them using Naïve Bayes classification to enhance the accuracy of classification methods. Another approach in the field of text mining for product classification was proposed by Sarwagi, Chakrabarti, and Godbole [10] known as cross mining. They dealt with overlapping semantics between sets of values and labels by using one to build a better classifier for the other if there was some similarity between their semantics. Cevahir and Murakami [5] proposed the use of machine learning techniques to identify product categories based on their title and description. Their idea was to use a combination of two neural network models namely deep belief nets and deep autoencoders. This model was trained on 150 million images categorized into 5 levels of hierarchy, but it required the use of graphical processing units (GPUs) to do so and attained 81% accuracy. Bhardwaj and Iyer [8] also opted for text-based classification and used different approaches such as the standard bag of words (BoW) method, word2vec, and support vector machines (SVM). The approaches mentioned so far are all text-based classification methods, and deal with challenges such as insufficient text, irrelevant text, incorrect text, and so on.

R. Kannan, Talukdar, Rasiwasia, and Ke [6] pointed that text alone cannot be used for active categorization of products, images need to be taken into account for better results. They performed supervised learning with logistic regression and then used a confusion driven probabilistic function

(CDPF) to identify what categories is their models most confused in. The model was then trained individually on these categories where it performs weakly. Another approach suggested by V. Gupta and H. Karnick [14] is to automatically assign tags to products based on their images. These tags them helped in easier categorization. They made the use of deep convolutional neural networks in order to retrieve features from images from products. once the feature map was obtained, they used K-nearest neighbors (KNN) to assign tags to similar kind of products belonging to the same cluster. L. Chang, F. Yang, and H. Yang also used a CNN based approach for classification as well as recommendation [15]. They constructed the AlexNet model using CNN and compared it with the baseline SVM. In order to perform recommendations, they used the feature vectors from the last layer of CNN to find similar products to show to the users. Another use of machine learning in computer vision to classify images was done by M. Hussain, J. Bird, and D. Faria [16]. They used CNN transfer learning for generic image classification. The CNN architecture model Inception-v3 was used to identify the time complexity and accuracy as compared to traditional neural networks.

4. ESSENTIAL NEURAL NETWORK CONCEPTS

4.1. Convolutional Neural Networks

A CNN constitutes of one input layer and multiple hidden layers. These hidden layers are comprised of convolutional layers, pooling layers, fully connected layers, and normalization/activation layers (ReLu). For even complex models, more layers can be added. Figure 4 shows the basic architecture of a CNN.

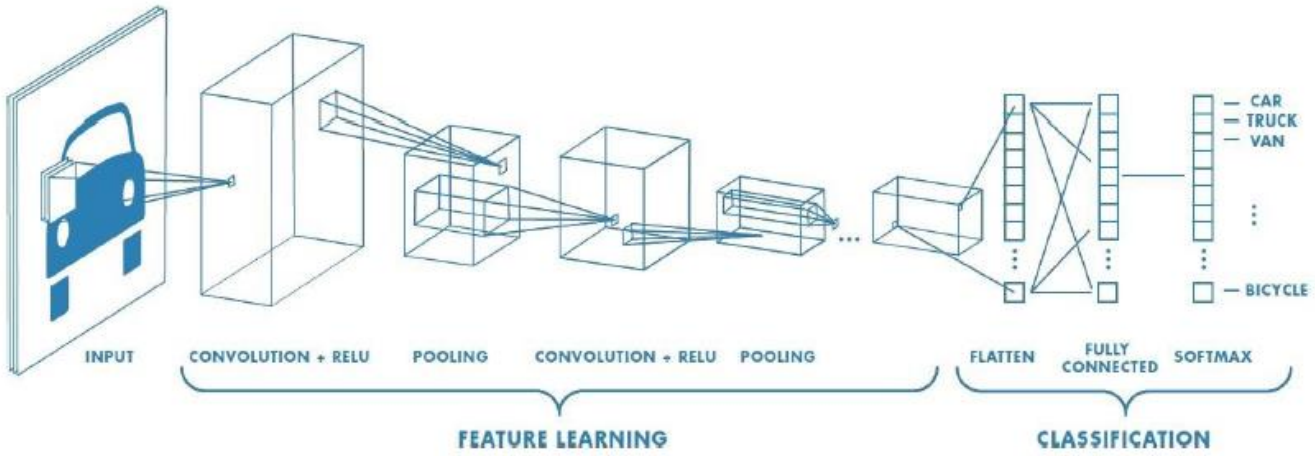

Figure 4. Typical CNN Architecture

CNNs work on the basis of linear algebra. Data and weights are represented in terms of matrix vector multiplications. Every layer of a CNN is responsible for incorporating a different set of characteristics for a set of images. For example, if the input to a CNN is the image of a face, then the initial layers will learn features of the face such as edges, curves, light spots, dark spots, lines, etc. The further layers in the network comprise of shapes and objects that can be identified in the face like mouth, nose, and eyes. The final layers then constitute of the features that make a face look like a face, meaning the collection and arrangement of the shapes and objects learnt till the previous layer to make a face. This way, a CNN functions by fragmenting the image into smaller portions called features and then matching those parts, not the whole image.

In order to derive these features of smaller portions of the image, a filter or convolution is used. It is generally a 3x3 grid that moves over the entire image thus aligning the feature with that part of the image. As the feature grid moves along the image, the pixels of the image portion in consideration are each multiplied with the corresponding feature grid pixels. Once the grid has finished moving over the image portion, all values obtained after multiplication are added up and divided by the total number of pixels in the grid. This value is then stored in the feature patch and this process continues for the entire

image.

Then comes the max pooling layer which involves a sliding window like concept that helps in reducing the size of the image stack. Once the convolutional layer helps extract features, max pooling helps to downsize the input that comes to it by selecting the most important features. It helps to avoid overfitting and decrease computational cost by reducing the number of parameters [17]. A window size needs to be decided to pick the best features from such as 2x2 or 3x3, also the stride needs to be specified. Stride refers to the pixels across which the window would slide or shift. Based on the window size and stride defined, the window slides over the image and the maximum value is saved that represents the best features from all the features obtained in the previous convolution step (Figure 5).

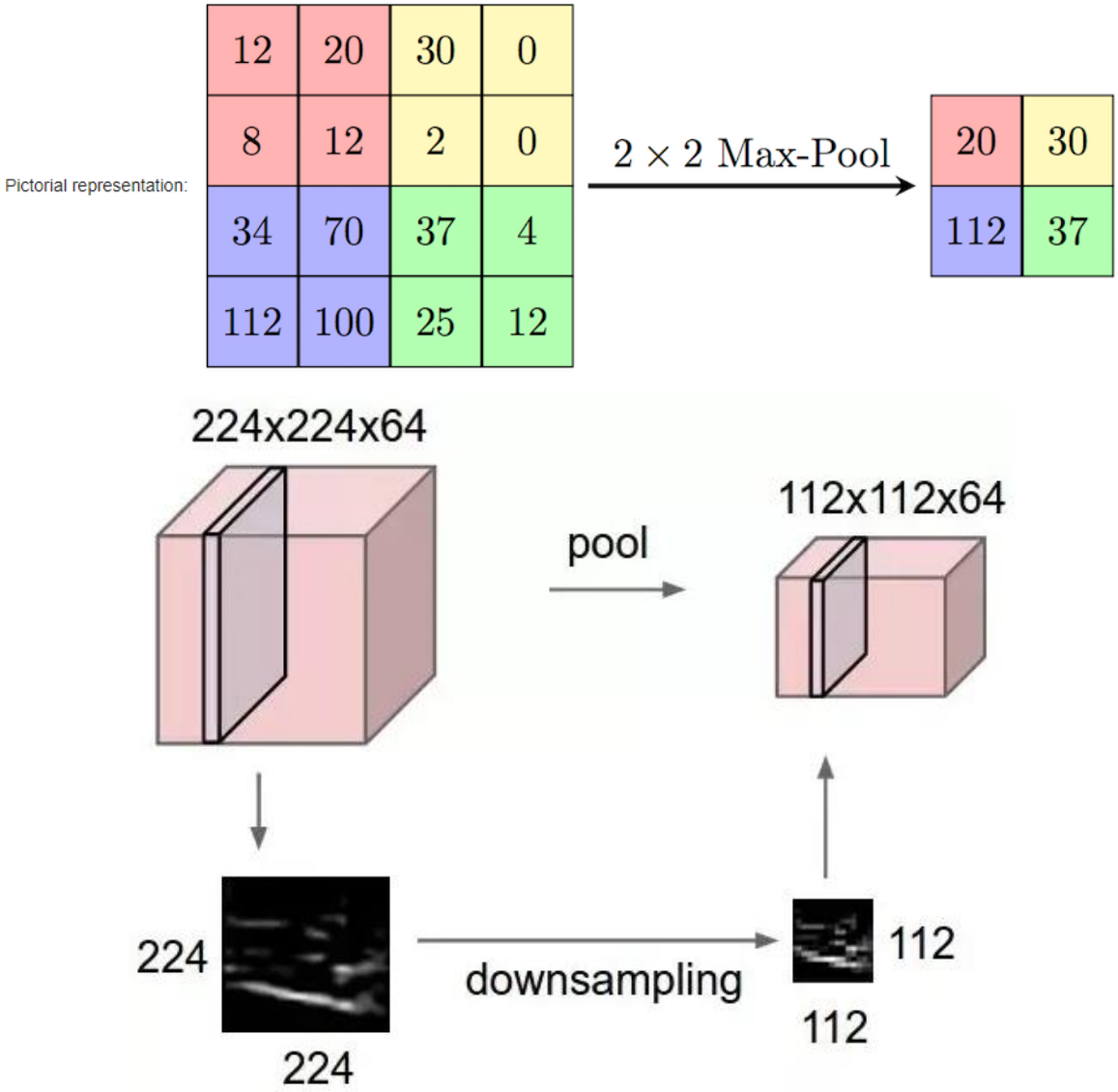

Figure 5. Down-sampling using Max Pooling
Source: Adapted from *[17]*

The next layer in the CNN is the normalization layer or ReLu. The term ReLu refers to Rectified Linear Unit and its task to convert all negative values to zero. It does this by applying an activation function such as *max(0, x)*. This process of normalization is continued for all the filtered images from the previous step. These layers are then stacked one after the other to make one layer's output become the next one's input. This process is known as deep-stacking.

After all these blocks of convolutional, ReLu, and pooling layers comes the final fully connected layer in the CNN. The class scores are computed at this fully connected layer. While the previous layers act as feature extractors, this final layer is the classifier. The values for the convolution layer and for the weights in the fully connected layer are acquired using the method of backpropagation. In any neural network, backpropagation is the technique with which the network uses the error at the end to adjust its weights and other values in order to minimize error and maximize accuracy.

4.1.1 CNNs vs Traditional Computer Vision

The reason behind the better performance of CNNs over traditional computer vision approach can be understood in the following way. For image classification, a model primarily performs two tasks – feature extraction and classification. Feature extraction refers to retrieval of surface information from the crude pixel values. This surface information needs to be able to seize the contrast between the various categories involved in the scenario. In this part of the pipeline, the end classes of images do not play a role. This means that feature extraction is an unsupervised technique to withdraw information from image pixels irrespective of what class the image belongs to. Once these features are obtained, a classification method is then trained to associate the images with their respective labels. This course of action towards classification suffers from some flaws. The first one being that the process of feature extraction cannot be modified on the basis of images and classes. Thus, if the selected features do not properly represent what it takes to differentiate between categories, the model performance and classification accuracy take a setback no matter how good the classification technique applied at the next step. A measure taken for this is to incorporate multiple feature extractors and then combine them to improve accuracy, but this leads to a lot of heuristics and parameter tuning in order to gain the desired level of accuracy. The other flaw with this approach is the lack of learning as time progresses. Machine learning is aimed towards performing human-like, the humanly approach to learn things is a gradual process. While deep learning works in the same direction of gradual learning with proceeding

epochs and iterations, the traditional methods adopt a more hard-coded feature extraction approach that does not let it change with the course of time. Deep learning, on the other hand, integrates the feature extractor and classifier into a combined entity that learns to extract features and distinguish classes as the iterations proceed. This task can be done using CNNs, in which there are sparse connections between the convolutional layers and neurons share parameters (Figure 6).

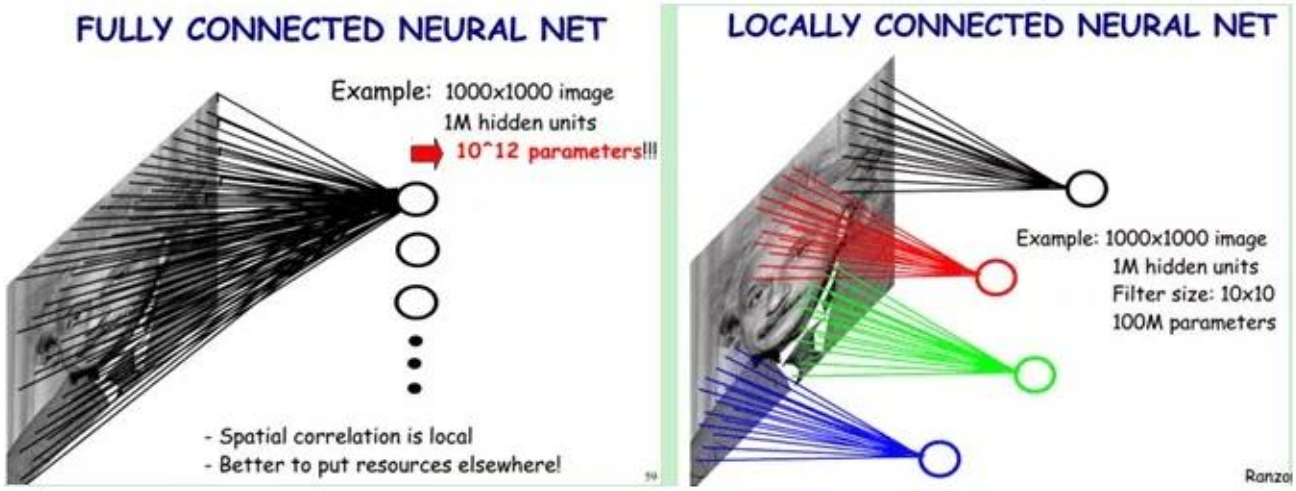

Figure 6. Local neuron connections in CNN

4.2. Dropout

With a large dataset such as ImageNet, there is a great possibility that a neural network training on it would tend to overfit on the data points. In order to avoid this, an approach could be to not use some units or neurons during forward or backward passes. This process of "ignoring" certain neurons to avoid overfitting is referred to as Dropout. These neurons that are supposed to be ignored during the training phase are chosen at random. It helps in getting more generalized and average results rather than those very specific to the given model only. Dropout is used in neural networks the same was as L1 or L2 regularization is used in logistic regression where a penalty or a bias is added to the losses so that the model does not learn interdependent weights and does not overfit. In this way, applying dropout is helpful to make the neural network learn features that can help in different subsets of other neurons also. The downside of using dropout is that it increases the number of iterations needed for the model in order to converge. But the upside is that the training time taken per epoch reduces. Figure 7 shows a neural network with all neurons used versus one with randomly chosen neurons ignored.

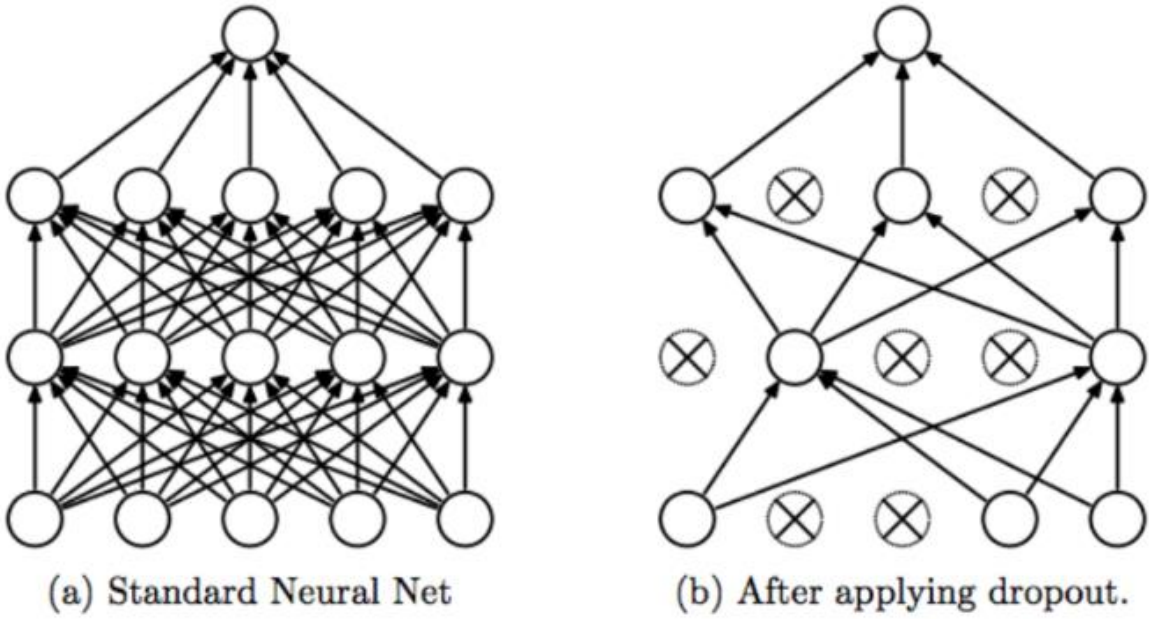

Figure 7. A random fraction of neurons ignored during dropout phase

5. TRANSFER LEARNING

Today we live in the age where data scientists and researchers in the field of machine learning are thriving for True Artificial General Intelligence (AGI). AGI refers to the intelligence of a machine that is capable of successfully executing any cognitive activities like a human would do. It is also referred to as strong AI or full AI. Experts believe that transfer learning could help us get one step closer to achieving this goal of making more and more human like in the way they think [18].

When human beings gain knowledge in a specific area, it is an implicit capability in them to apply that knowledge to other areas [18]. For example, when they learn how to ride a bicycle, they'd use that knowledge to learn how to ride a motorbike; and then that knowledge to learn how to drive a car. This implies that we do not learn everything from the very beginning, we transfer learnings gained from one task to another. The more related the new task is to an old one that we have learned in the past, the easier it becomes for us to excel at it using our previously gained knowledge.

While experts in the field of machine learning have been working continuously to make it as close to human learning as possible, this practice of knowledge reusability has not been widely implemented. The traditional machine learning algorithms had been designed to work in isolation. These conventional models are structured for a particular task and they have to be rebuilt from scratch when the input data or the feature space distribution changes. Transfer learning is an approach that aims at changing this by devising techniques that could be applied to use knowledge gained from a source task and re-use it to quicken the learning of a related target task (Figure 8).

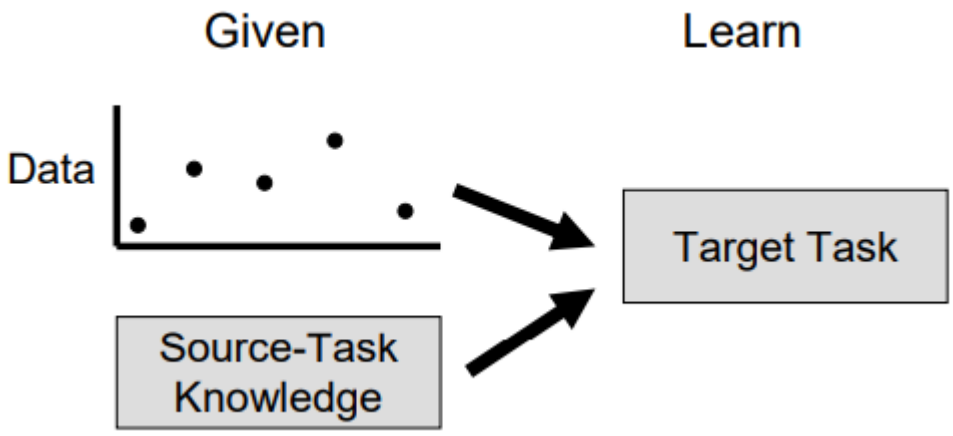

Figure 8. Use of given and extra knowledge by transfer learning

Source: Adapted from [19]

5.1. History of Transfer Learning

The conventional machine learning algorithms predict results for test data based on their training on the labeled or unlabeled data. The algorithms that use labeled data for predictions are called supervised algorithms while the ones that use unlabeled data are known as unsupervised algorithms. There also are cases when there is very little labeled data and a good classifier cannot be built on such small dataset. Therefore, the model uses a large amount of unlabeled data and whatever little labeled data that is available. This type of algorithms is known as semi-supervised algorithms that work on a mix of labeled and unlabeled data. For all these algorithms and their variations, a lot of research and study has been done with one assumption constant among all studies – that the data belongs to the same distribution [20]. Contrary to this, transfer learning provides the freedom to use datasets from different distributions. For example, in a real-life scenario, we might have a model trained on data from a university's web pages to classify documents that have been labeled manually. Transfer learning allows the information gained from training on this data to be used for a newly created website for some different domain.

The idea of transfer learning came into being from the 1995 Neural Information Processing Systems (NIPS) workshop called "Learning to Learn: Knowledge Consolidation and Transfer in Inductive Systems". This workshop provided the initial motivation for people to start researching in this field and ever since, these phrases "learning to learn", "inductive transfer", and "knowledge consolidation" are used interchangeably with transfer learning. Transfer learning is quite connected to a concept called multi-task learning, although they are not the same [21]. Multi-task learning refers to a framework where a model would learn more than one tasks at the same time even if the tasks are not similar. It does so by finding out the common features between the tasks and leverages this connection to benefit both tasks. After NIPS initiated the process, then the Broad Agency Announcement (BAA) 05-29 of Defense Advanced Research Projects Agency (DARPA)'s Information Processing Technology Office (IPTO) 2 gave a new definition for transfer learning in 2005. They defined transfer learning as "the ability of a system to recognize and apply knowledge and skills learned in previous tasks to novel tasks". Based on this definition, the goal of transfer learning was to retrieve information from one or more tasks known as the source task(s) and apply that knowledge to a target task. Further, from 2010, this field of transfer learning picked up more pace and started being used in several fields, most prominently in that of data mining and machine learning, and their areas of application.

5.2. Motivation for Transfer Learning

One of the primary stimuli for transfer learning can be pointed towards the fact that the core requirement for a supervised deep learning model to perform well and solve complicated tasks is data, lots of labeled data. By labeled data in this scenario, it means that images must have their respective category labels. Firstly, getting a huge amount of data in itself is not a very easy task owing to the dynamic loading of product images on ecommerce websites and privacy concerns as well. Also, labeling all these images is a very time taking process. The other key motivation is the isolation characteristic of most deep learning models. It means that these models are built for specific tasks, and in those tasks, they provide never seen before accuracy. But the moment the dataset is changed, they suffer from noteworthy losses even if the new dataset is similar to the original one that the model was trained upon. The requirement of a whole lot of labeled data and the absence of model reusability option with traditional deep learning methodologies open up the window for transfer learning.

Transfer learning, that works on a model already trained on a huge dataset and leverages that knowledge by applying it to a smaller data while maintaining accuracy, solves the above purpose. It bypasses the requirement for lots of data and the necessity to train from scratch by using the features and learning from another related yet bigger dataset.

5.3. Transfer Learning Explanation

Transfer learning aims at improving the performance of the target task by using information acquired from the source task. This betterment of learning is done in three ways (Figure. 9). The first way is the improvement in the initial performance of the target activity by using only the derived knowledge before learning anything new from the current dataset. This gives a jump start to the learning process as compared to the traditional model that starts from zero. The second way is the improvement in performance of the target task is by reducing the total time taken for the target model to learn. Using the transferred information from the source, the time to learn can be greatly reduced as compared to the time it would take for this agent to learn anew. The third way the learning procedure is enhanced is measured by its final performance using the desired unit like accuracy, time taken, losses, etc. If the metric chosen to measure final performance is accuracy, transfer learning is supposed to improve the performance of the source model by increasing its accuracy s compared to the accuracy that would have been attained if the model was trained from scratch.

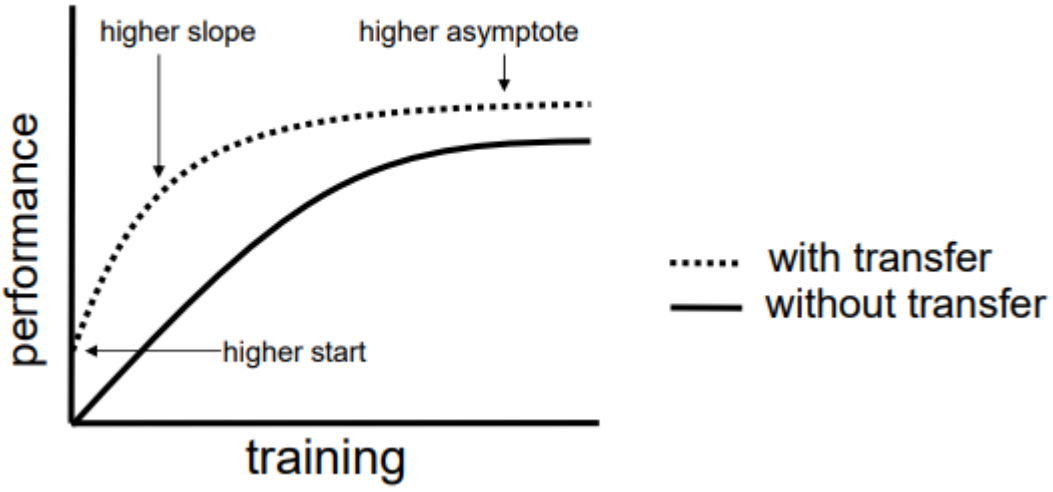

Figure 9. Three ways transfer learning could improve learning
Source: Adapted from [19]

To apply transfer learning to a given problem, one needs to answer three questions namely what to transfer, when to transfer, and how to transfer.

"What to transfer" is the first step of the transfer learning process that deals with the part(s) of knowledge that can be transferred. It is most critical to identify this part in order to successfully utilize information gained from source to target. This is done by identifying the parts of knowledge at the source end and the ones that are common between the source and target.

"When to transfer" deals with the cases in which knowledge should or should not be transferred. We say should not be transferred because it could happen in certain scenarios that using information from another model might actually degrade the performance of the target task. This situation is called as "negative transfer", and it is still an open issue to preemptively identify this case. But we need to be careful to use transfer learning in cases only where it improves the performance.

"How to transfer" refers to the ways in which knowledge shall practically flow from source to target across domains and tasks once the what and when have been figured out. It includes identifying the parts of existing algorithm that can be used as is, the ones that need to be made, and implementing those changes to improve performance of the target algorithm.

To understand transfer learning, it is important to note that it is different from both traditional machine-learning as well as multi-task learning. We shall look at the differences one by one here.

Traditional ML, as mentioned before, is task-specific while transfer learning is not (Figure. 10). A model

is built and tuned to suit a particular task in isolation without preserving any information that could be used for another model. In case of transfer learning, information such as weights, features, etc. is saved from the source model and is used to train new models even if they have lesser data available. For example, there's a task T1 to find objects in a restaurant's images. For the dataset containing these images, we train a model and tune its parameters and the model performs well on the test data as well. The point to be noted is that the test images belong to the same domain, i.e. restaurant images. Now if we had another task T2 which required us to identify objects in a park, preferably, the current model should have been able to do so. But it does not, because the current model does not generalize well owing to its bias towards the original data and domain.

In the case of transfer learning, we can save the features and weights from task T1 and use them for T2 provided the input data for T1 was more than that for T2. For computer vision related problems, some low-level features like shapes, edges, corners, intensity, etc. can be saved and used for multiple tasks. This knowledge from one task acts as a supplementary input for another task along with the new task's own data and learning.

When it comes to distinguishing transfer learning from multi-task learning, though the former uses some ideology of the latter, the two are different in terms of operation (Figure. 11). Multi-task is the approach of learning various tasks simultaneously with source and target tasks playing symmetric roles. This means that there is no delegation of a task as source or the other as target, they are all equal and are supposed to be learnt at once by the agent. On the other hand, in transfer learning, the source and target tasks are clearly appointed, and the agent while learning from the source task has no information of the target task. Based on this explanation, it can be deduced that dealing with a multi-task learning scenario with a transfer learning method is possible, but the vice-versa is not.

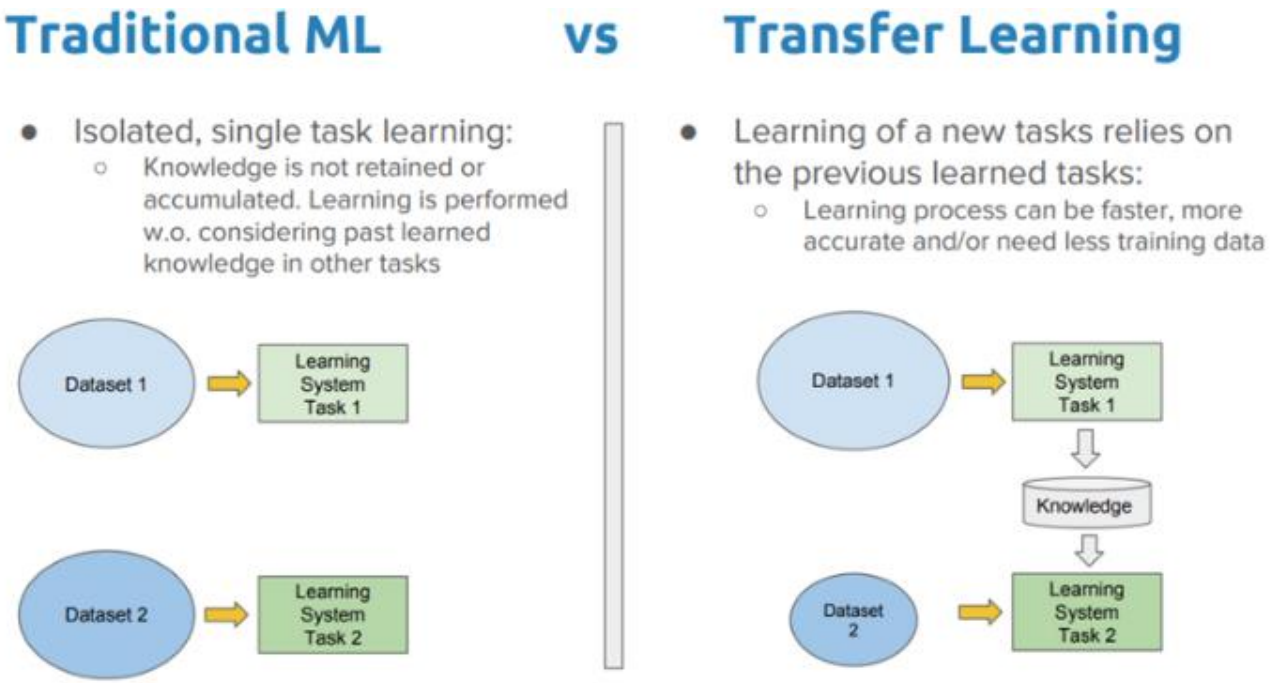

Figure 10. Traditional ML vs Transfer Learning
Source: Adapted from [18]

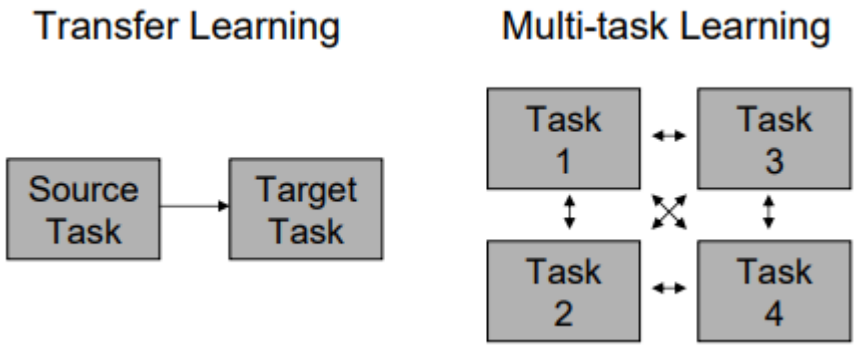

Figure 11. One direction information flow in transfer learning, free flow of information in multi-task learning
Source: Adapted from [19]

5.4. Notations and Definitions

Before we delve deeper into transfer learning, we shall discuss the notations and definitions used in this survey to get a better understanding of the transfer learning framework.

A domain, D, is defined as constituting of two elements – a feature space χ, and marginal probability P(X) where $X = \{x_1, x_2, \ldots, x_n\}$, $x_i \in \chi$. So, it can be mathematically represented as **$D = \{\chi, P(X)\}$**. This can be

explained with the help of an example. Suppose the task at hand is to classify documents with every term considered a binary feature, then χ is the feature space of all these term vectors, x_i is the i^{th} term vector for some documents, and X refers to a specific learning sample [20].

The next definition is of a task T, that can be defined as an ordered pair of the label space ϒ, and an objective function *f(.)*. So, a task can be denoted mathematically as **T = {ϒ, *f(.)*}**. This function consists of pairs $\{x_i, y_i\}$ where $x_i \in \chi$ and $y_i \in \Upsilon$ and is used to make predictions of the respective label $f(x_i) = y_i$ for an instance x_i. In probabilistic terms, this objective function can be defined as P(ϒ|X). Following the above used example for classifying documents, ϒ would be the set of all labels, i.e. true or false in this case. So y_i would be able to take two values – true or false.

Having defined domain and task, we now give the formal definition of transfer learning as follows. Let Ds be the source domain and Ts be the respective source task. Also, let Dt be the target domain and Tt be the target task. Then transfer learning's goal in this case should be to use the information gained from Ds and Ts in order to enable us to learn the target conditional probability distribution P(Yt|Xt) in the domain Dt where Ds≠Dt and Ts≠Tt [22]. If the source and target domains are same, it just becomes a traditional machine learning problem. When the domains are different, it gives rise to four scenarios as follows:

1. Xs≠Xt => this implies that the source and target domains have different feature spaces. In the document classification example, this would imply that the documents are written in different languages.
2. P(Xs) ≠ P(Xt) => this implies that the source and target domains have same feature spaces but different marginal probability distributions, meaning the documents talk about different topics.
3. Ys≠Yt => this refers to different label spaces for the source and target domains, i.e. the source document being classified into binary classes whereas the target document being classified into any number of classes other than two.
4. P(Ys|Xs) ≠ P(Yt|Xt) => this refers to class imbalance leading to different conditional probability distributions in source and target domains. This corresponds to the case where source and target documents are not balanced in terms of their user-defined classes.

5.5. Transfer Learning Strategies

On the basis of domain, tasks, and availability of data, transfer learning can be categorized into three different sub-settings (Figure. 12) namely inductive transfer learning, transductive transfer learning, and unsupervised transfer learning.

1) In inductive learning, the source and target domains are the same, but the source and target tasks must differ from each other. The inductive biases of the source domain are utilized to improve the performance of target task. It requires some amount of labeled data to be present in the target domain. Depending on the availability of labeled data in the source domain, it can further be categorized into two parts. If the source domain has labeled data, it becomes similar to multi task learning, and if not, it is like self-taught learning.
2) In the case of transductive learning, the source and target domains are different, but the tasks are same. Here, the source domain consists of a whole lot of labeled data while the target domain does not have any. Based on the difference in respective feature spaces or marginal probabilities, transductive learning can further be classified into subcategories.
3) Unsupervised learning is similar to inductive learning in terms that source and target domains are similar, and the source and target tasks are different but related to each other. It lays attention on unsupervised learning in the target domain, and as the name suggests, none of the domains contain labeled data. Table 1. summarizes the above information.

TABLE 1
Settings for Transfer Learning Strategies

Transfer Learning Strategy	Related Fields	Source and Target Domains	Source and Target Tasks	Source Domain Labels	Target Domain Labels	Tasks
Inductive transfer learning	Multi task learning	Same	Different but related	Available	Available	Classification, Regression
	Self-taught learning	Same	Different but related	Unavailable	Available	Classification, Regression
Transductive transfer learning	Domain adaptation, sample selection bias, co-variate shift	Different but related	Same	Available	Unavailable	Classification, Regression
Unsupervised transfer learning		Different but related	Different but related	Unavailable	Unavailable	Clustering, Dimensionality Reduction

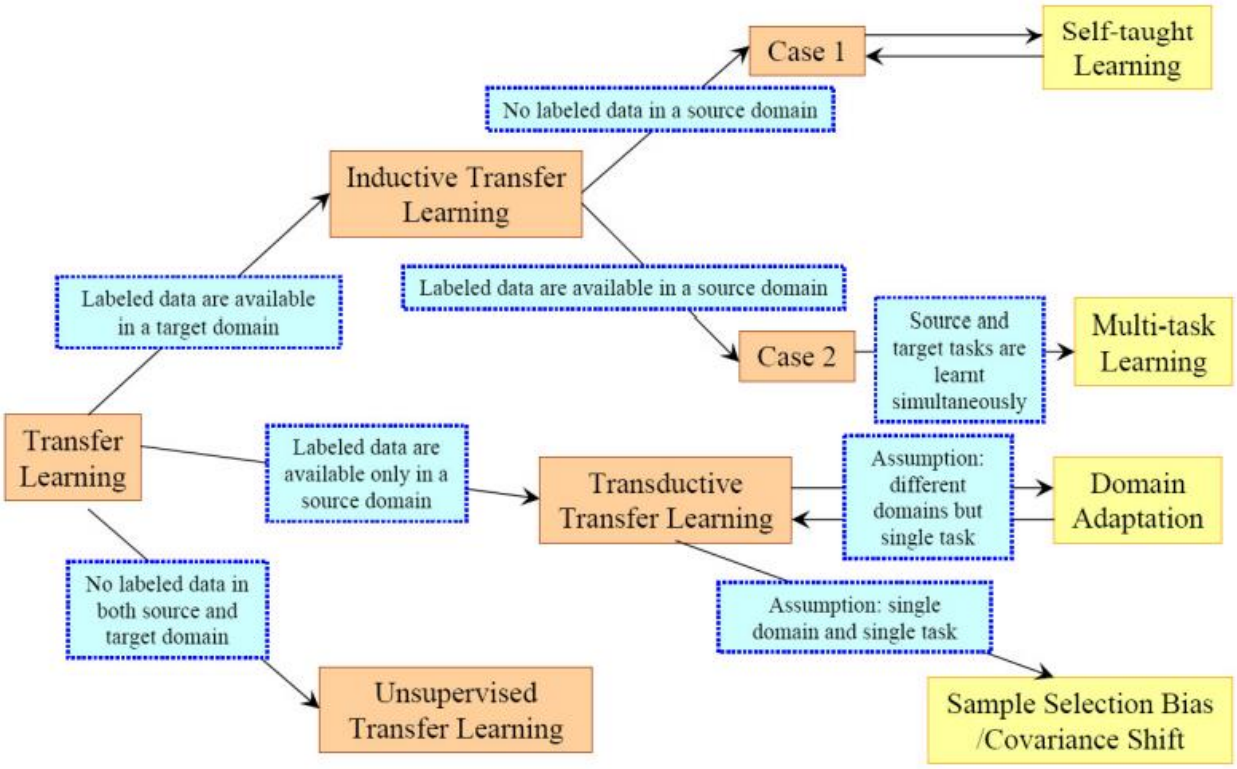

Figure 12. Transfer Learning Strategies
Source: Adapted from [20]

As discussed earlier, when to transfer, what to transfer, and how to transfer are the three questions that need to be answered when applying transfer learning to any problem. The above categorization answers when transfer learning can be applied under what data availability circumstances. Next comes what information could be transferred across these categories. The answer to what to transfer from source to target domains in the above mentioned three settings can be deduced on the basis of the following approaches. The first approach is referred to as “instance transfer” which takes into account reusing parts of data from source domain for the target domain. Under majority circumstances, the data from the source domain cannot be used as is for the target domain, either certain instances can be used, or the data needs to be modified in some way. These modification techniques consist mainly of re-weighting and importance sampling. The second approach is “feature-representation transfer” that considers the transfer of knowledge by encoding it into the learned feature representations from the source domain. This approach aims towards finding good feature representations that can be used for the source domain for the target domain in order to prune down errors as much as possible. The third approach to transfer knowledge is known as “parameter transfer” which makes an assumption

that the source and target models either share some parameters or a prior distribution of the hyperparameters. In this case, the information to be transferred is encoded into priors or shared parameters and passed along from the source to target domains. The fourth and final approach towards answering what to transfer is known as the “relational-knowledge transfer”. It deals with data that is not independent and identically distributed (non-IID). This means that each of the data points shares some connection with the other data points, for example in a social network. This approach assumes that there is some similarity in the relationship shared by data points of the source and target domains and the knowledge to be transferred is this relationship. Table 2. depicts the relationship between the various transfer learning strategies and approaches.

TABLE 2
Transfer Learning approaches used in different settings

	Inductive transfer learning	***Transductive transfer learning***	***Unsupervised transfer learning***
Instance transfer	√	√	
Feature representation transfer	√	√	√
Parameter transfer	√		
Relation knowledge transfer	√		

Form the above table, we can see that inductive transfer learning is the most widely researched strategy in the field of transfer learning and has been explored for all approaches. Unsupervised transfer learning, being a relatively newer research area has only been studies in the context of feature representation transfer. Also, we can see that it is only the feature representation transfer approach that has been studied upon with all three strategies while parameter transfer, and relation knowledge transfer have only been researched along with inductive transfer learning.

The approach used in this paper is based on feature-representation transfer based inductive transfer learning that we shall be talking more of in the upcoming sections.

6. PROPOSED APPROACH

This paper proposes to use transfer learning for the effective classification of ecommerce products in their respective categories. It also compares the performance of traditional CNN versus that of transfer learning in terms of time taken and accuracy. The input to the proposed model is an image of any object that the customers would want to buy on the ecommerce platform.

6.1. Objective and Motivation

The field of transfer learning, though not completely new, is still an unplumbed one. Also, while work has been done to perform image classification using transfer learning [16], using transfer-learning - based image classification to categorize products is an untapped amalgamation of use-case and solution. This novel fusion of the newness of transfer learning and explosiveness of ecommerce websites with a plethora of products is the key motivation for this project.

Furthermore, ecommerce websites do consist of millions of products as a whole, but the number of products for each category are still limited. To train any machine learning model on these small number of images for individual categories leads to underfitting and inefficiency in terms of performance. The ability of transfer learning to augment data and use information saved from a previous model helps overcome this problem.

6.2. Dataset and Models Explained

6.2.1. Source Dataset and Models

There are multiple transfer learning models that have been trained from scratch over millions of images and can be readily used by fine tuning certain layers and parameters / hyper parameters. For predictions of images, a commonly used dataset for the source model is ImageNet. It is a database consisting of millions of images belonging to thousands of categories [23]. This tedious task of labeling images into all these categories is a result of years of hard work done at Stanford. The inspiration for the ImageNet project was the growth in the field of computer vision and the requirement of more image data. After it was developed, ImageNet started a yearly challenge by the name of ImageNet Large Scale Visual Recognition Challenge (ILSVRC) for the assessment of object detection and image classification algorithms (Figure. 13).

Figure 13. ImageNet Challenge based on the ImageNet Database

The algorithms that won the ImageNet challenge over the years were majorly the ones that used Convolutional Neural Networks. The very first deep learning model to bring about a significant change in the accuracy over the ImageNet dataset was AlexNet. It used 5 convolutional layers followed by 3 fully connected layers. This network changed the activation function to ReLu as opposed to Tanh or sigmoid that were most popular at the time to handle the non-linear part. The advantage being that ReLu takes much lesser time to train. AlexNet also solved the problem of overfitting by using a Dropout layer after every fully connected layer. The next model that had considerable impact on accuracy over the ImageNet dataset was the VGG-16. This project uses VGG-16 as the pre-trained source model. Its architecture comes from Oxford's Visual Geometry Group (VGG). It betters the AlexNet model by substituting the large filters with multiple 3x3 kernel-size filters stacked consecutively. This helps in increasing the depth of the network so that it can learn more complex features more easily while still maintaining the cost at a lower end. Another advantage of using multiple smaller filters is their capability of capturing even the finer properties of images and thus producing better results. After VGG-16, other models like GoogleNet and ResNet were also developed, but the fact that VGG-16 converges and trains quickly made it choice of source model for this project. Also, VGG-16 is conceptually simpler and easier in terms of implementation which makes it a more popular choice in general.

The VGG-16 architecture consists of 16 layers including the convolution layers and the fully connected layers (Figure. 14). This network was built upon the ImageNet dataset by K. Simonyan and A. Zisserman [24].

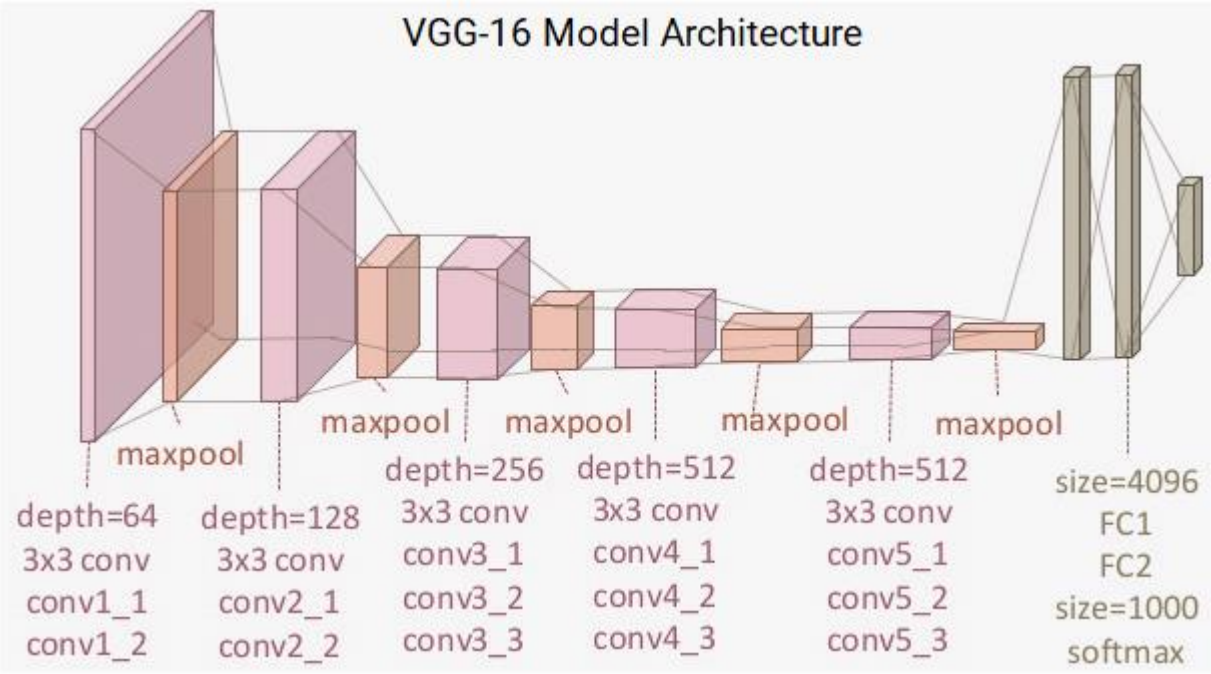

Figure 14. VGG-16 Model Architecture
Source: Adapted from *[18]*

6.2.2 Target Dataset and Models

The structure of the present ecommerce websites is quite branched and complex. The home page consists of different categories and within those categories are sub-categories and so on. The requirement is to collect images for these categories and their sub-levels in a labeled fashion. Labeled here implies that images of a particular category need to be in the folder with the category name. Also, these product images are loaded on the website dynamically using ajax calls, therefore they could not be accessed directly using their ids in the html script. Collecting these images individually would have been a very time taking task, thus to get these images in the specific directory structure, a web crawler was written to automate the process of image collection in the particular folders. Using the image crawler, around 17,500 product images belonging to 5 categories were collected from an ecommerce platform for the target dataset. Of these, we use around 900 samples per class as test data and rest we keep for our training and validation part. Further explanation of the language, tools, and process used to write the image crawler and automate the downloading of images is explained in section 6.3, “Tools and Technologies Used”. The dataset is prepared in the following structure data-> train -> appliances-> train_images_of_appliances where appliances is one of the categories of data, and this structure is

followed for all categories. Similarly, the test folder is prepared. Once all the data is downloaded and assigned to respective directories that act as labels for the images, it is then ready to be fed as input to the target model.

For the target model, the initial layers containing the weights and features are used of the VGG16 model trained on the ImageNet dataset. Figure 14 shows the 13 convolution layers in the network each using 3x3 convolution filters. Also, max pooling is used in order to down sample the input and at the end are two fully connected layers with 4096 neurons in each layer. Finally, there is a dense layer consisting of 1000 units, with each unit being a representative of each category in the ImageNet database. For the target model though, we do not require the last three layers, the two fully connected and a dense one. these end layers are the classifiers, and for our dataset and classes, we use our own fully connected and dense layers. The first five chunks consisting of the convolution layers, activation layers, and max pooling layers are used as feature extractors for our target model. We use freezing and fine tuning for the pre-trained model to suit our purpose. Freezing means fixing weights for the layers, so they are not updated during backpropagation, while fine tuning refers to the phenomena where we tune the weights of the layers. Figure 15 shows the difference between the two and explains which one to use under what circumstances. For our model, we discard the top layers of VGG-16, i.e. we do not use the final block consisting of fully connected and dense layers. As for the initial five blocks constituting convolutional and max pooling layers, we freeze them to use them as feature extractors. For the final layers, we add a flatten layer to flatten the outputs obtained from the convolutional layers. Then we add a dense layer with ReLu activation function, a dropout layer to avoid overfitting, and two more dense layers with ReLu and sigmoid activations respectively. This model with feature extractor from VGG-16 and fully connected layers to classify products into their categories is ready to be used for the desired goal.

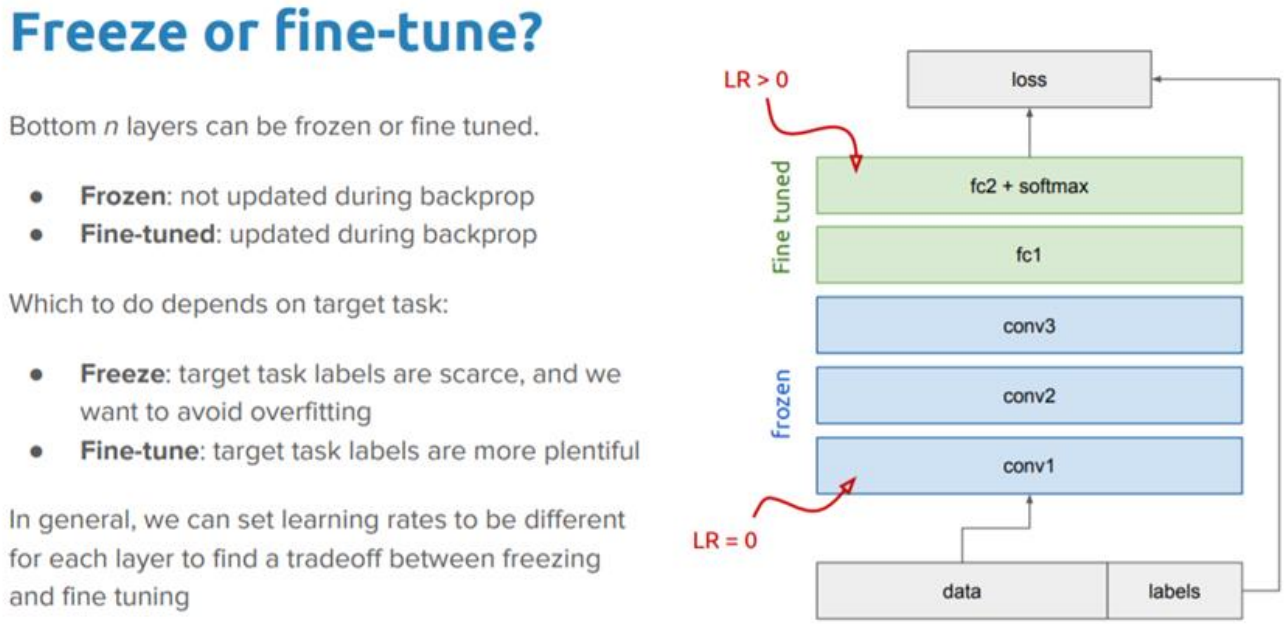

Figure 15. Freezing versus Fine Tuning
Source: Adapted from *[18]*

6.3. Tools and Technologies Used

- To accomplish the desired outcome, Python 3 was used as the programming language utilizing libraries like numpy, pandas, SciPy, PIL, and matplotlib for plotting graphs.
- Jupyter Notebooks by Anaconda v. 1.9.4 was used as the platform to code in python since it provides the ability to run individual cells independently. This helps not only in easy debugging, but also saves state that helps in running the program from a specific point and not re-run the hole code all over again each time. This helps save a lot of time.
- Keras with Tensorflow backend is used as the library for the VGG-16 model. Keras is a high-level API that helps with the building and training of neural networks [25]. Tensorflow is an open source artificial intelligence library that helps in numerical computations using data flow graphs.
- Selenium was used to capture the dynamically loaded images on the ecommerce website in order to collect data. The images on the website load through ajax calls and cannot be captured directly using their html script ids. Therefore, to scrape this data, selenium, the automation tool was used in conjunction with python.
- BeautifulSoup is library in python that helps in retrieving data out from HTML and XML files. It provides several ways of navigating, searching, and modifying the parse tree obtained from the HTML/XML. In this project, version 4 of BeautifulSoup was used to obtain image URLs from the XML derived using Selenium. These URLs were then used to download the actual images to the desired folder.

6.4. Experiments and Results

The system configurations for this project involved a machine with an 8GB RAM, an i3 core processor, and an AMD Radeon Graphical Processing Unit (GPU). The gathered data could be stored locally since it did not involve millions of images for this project. Still, as an efficiency measure, Cassandra was used to store data and a python connection to Cassandra was created to use the stored data.

This project uses transfer learning with freezing and fine tuning. The first phase of the project comprised of drawing a comparison between traditional CNN and transfer learning. A basic CNN model with 5 blocks of convolutional, activation, and pooling layers was used followed by fully connected layers. The output of feature extractor layers was first flattened and then fed to a dense layer, then dropout was applied to avoid overfitting. Thereafter, another dense layer with sigmoid activation is used. The model is compiled using binary cross-entropy as the loss function and adam as the optimizer. On training the model on a subset of data consisting of 3039 images only, it is observed that the time taken is more than 3 hours to achieve 79% accuracy. There are two major problems with this model. First, the time taken for such less amount of data is quite a lot and as the size of data increases, this time will only go higher. Secondly, in Figure 16 that shows the graphs for accuracy and loss with increasing epochs for a basic CNN, it can be seen that the model tends to overfit on the training data which leads to almost 90% train accuracy while the test accuracy gets sinking low as the epochs proceed.

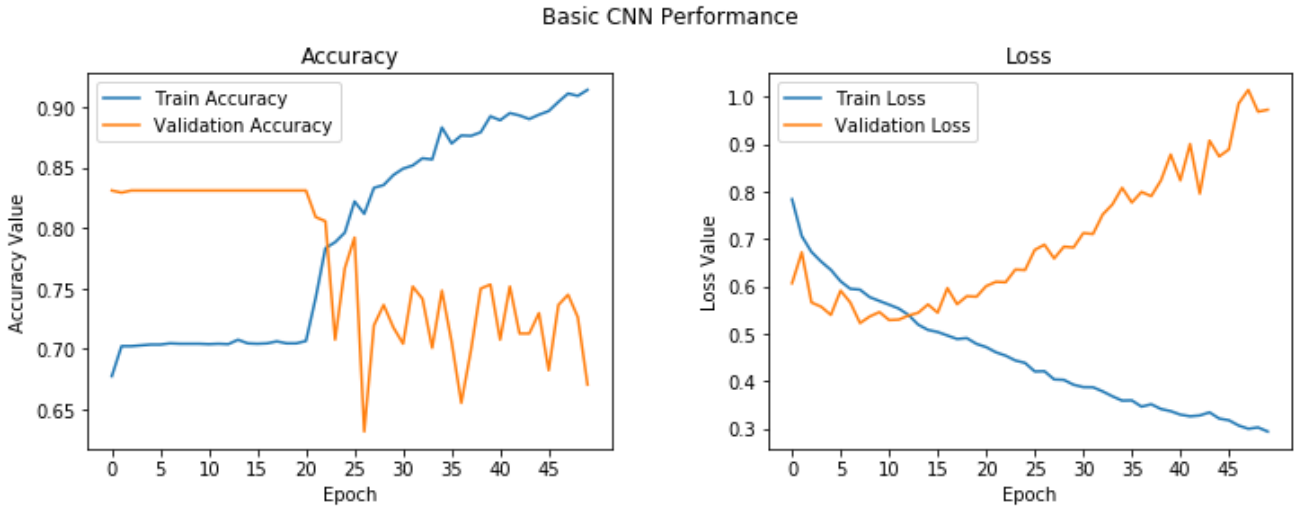

Figure 16. Accuracy and Loss graphs for a Traditional CNN

Using transfer learning on the other hand gives and average accuracy of 85% while taking only 16.96 minutes, which is drastically better as well as faster as compared to the traditional CNN. Figure 17 shows the accuracy and loss graph for transfer learning and it can be seen that while the model shows some spikes in the validation data, it majorly is because this comparison was done on a subset of data. With more data, the graphs tend to get smoother.

Figure 17. Accuracy and Loss graphs for Transfer Learning

The next phase of the project involved applying transfer learning to various product images from the ecommerce website and testing the model. This could be done in two ways. The first one was to train a model on all the images at once and train it for multi-class classification. This method gave very poor accuracy of 35.9% and high loss.

The other technique is what we call “one versus all”. This refers to taking samples from one category as the first class and combined, but equivalent number of samples from all other categories together as the other class. This leads to a binary classification like situation where the model learns to classify one category of products against those of all other categories. To implement this, the bottleneck features of each class were obtained individually and a separate model for each category was built. The feature extracting layers of the VGG-16 model were locked and the classifier layers were fine tuned. Dropout was used in the process to avoid overfitting and Adam was used as the optimizer with a learning rate of $1e^{-5}$. Figure 18 shows the test results and the accurate classifications from the trained appliance model. It shows how the appliance model data generator assigns 0 to appliance class and 1 to all other

images. On running the predict function, it can be seen that the model returns class 0 for the appliance image while it returns class 1 for all other test images. In the same way, models for each of the remaining categories was trained and tested for images from all classes. An average accuracy of 89% was observed among all models which.

```
In [88]: appliances_train_generator.class_indices

Out[88]: {'appliances': 0}

In [113]: predictions_app = model.predict(features_app)
          print(predictions_app[0][0] < 0.5)
          model.predict_classes(features_app)

          True

Out[113]: array([[0]])

In [114]: predictions_furn = model.predict(features_furn)
          print(predictions_furn[0][0] < 0.5)
          model.predict_classes(features_furn)

          False

Out[114]: array([[1]])

In [115]: predictions_elec = model.predict(features_elec)
          print(predictions_elec[0][0] < 0.5)
          model.predict_classes(features_elec)

          False

Out[115]: array([[1]])

In [116]: predictions_toys = model.predict(features_toys)
          print(predictions_toys[0][0] < 0.5)
          model.predict_classes(features_toys)

          False

Out[116]: array([[1]])
```

Figure 18. Appliance Model Test Results

7. CONCLUSION AND FUTURE WORKS

In this research, a project was implemented to apply transfer learning for the classification of ecommerce products into various categories. Both these fields of ecommerce and transfer learning are growing immensely day by day and the combination of these two is an untapped area as of now. In these ecommerce websites, there are millions of products, both from the website itself and from third party sellers as well. With so many products coming from various sources, it becomes difficult to establish a standard taxonomy. Thus, if there is a way to automate the process that classifies the products to their correct categories and can check for products that have been misclassified, it could prove very useful for both the website and users. It would be beneficial for the ecommerce website to have its products arranged in an organized way in the accurate categories so that users can navigate to the products through category pages easily. The better a user's experience on the website, the better it is for the website. For this to be implemented practically, a dataset of 17,500 product images belonging to 5 categories was collected from an ecommerce website. A web crawler was written to download images and collect the dataset. On that dataset, it was first tested whether transfer learning would improve performance over traditional CNNs. In an experiment done to compare the two models over the same set of data, it was found out that both had reasonably good accuracy, but the basic CNN model took a little more than 3 hours to train on the data while the transfer learning model took merely 16 minutes. After this, models for each category were built that showed remarkable results where each one was trained by keeping one category as one class and images from all other categories as the other class.

In the future, this algorithm can be applied to deeper levels of categories to get even better product taxonomy. For example, currently the category electronics was taken as a whole. To improve this, sub categories from within electronics such as laptops, TVs, tablets, etc. can be considered as separate entities and individual models can be built for each one of them to classify products into more granular chunks. Also, this approach can be combined with near matching to act like a recommender. So, once a product has been classified into its respective category, an extension of the current algorithm can be introduced that finds from within the database similar products since it already has product and category information. This, if implemented in real time can be very useful to display similar products to the users once they land at the product of their choice to provide them with more relevant options for the kind of item they are looking for.

BIBLIOGRAPHY

[1] K. Fiore, "How In-Store Retail Experiences Push Customers To Shop Online," Forbes, 2018. [Online]. Available: https://www.forbes.com. [Accessed 3 September 2018].

[2] S. Gottipati and M. Vauhkonen, *E-Commerce Product Categorization.*

[3] B. Researchers, "Product List: Category Insights," [Online]. Available: https://baymard.com. [Accessed 9 January 2019].

[4] I. Partalas and G. Balikas, "e-Commerce product classification: our participation at cDiscount 2015," June 2016.

[5] A. Cevahir and K. Murakami, *Large-scale Multi-class and Hierarchical Product Categorization for an.*

[6] A. Kannan, P. P. Talukdar, N. Rasiwasia and Q. Ke, *Improving Product Classification Using Images.*

[7] P. Radhakrishnan, "What is Transfer Learning?," 17 November 2017. [Online]. Available: https://towardsdatascience.com. [Accessed 16 October 2018].

[8] A. Bhardwaj and S. Iyer, "Large-scale product classification via text and image-based signals using a fusion of discriminative and deep learning-based classifiers," in *O'Reilly Conference*, San Jose, 2016.

[9] R. Agrawal and R. Srikant, "On Integrating Catalogs," *WWW '01 Proceedings of the 10th international conference on World Wide Web,* pp. 603-612, 1 May 2001.

[10] S. Sarawag, S. Chakrabarti and S. Godbole, "Cross-training: learning probabilistic mappings between topics," Washington, D.C., 2003.

[11] F. Qiu and J. Cho, "Automatic Identification of User Interest For Personalized Search," Los Angeles.

[12] R. Fishkin and M. Staff, "How Search Engines Operate," [Online]. Available: https://moz.com. [Accessed 9 November 2018].

[13] Z. Kozareva, "Everyone Likes Shopping! Multi-class Product Categorization for e-Commerce," in *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL*, Denver, 2015.

[14] V. Gupta and H. Karnick, "Automatic tagging and retrieval of E-Commerce products based on visual features," in *Proceedings of NAACL-HLT 2016*, San Diego, 2016.

[15] L. Chen, F. Yang and H. Yang, "Image-based Product Recommendation System with Convolutional Neural Networks".

[16] M. Hussain, J. J. Bird and D. R. Faria, "A Study on CNN Transfer LEarning for Image Classification," Birmingham.

[17] "Max-pooling / Pooling," [Online]. Available: https://computersciencewiki.org. [Accessed 24 November 2018].

[18] D. Sarkar, "A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning," 14 November 2018. [Online]. Available: https://towardsdatascience.com. [Accessed 25 February 2019].

[19] L. Torrey and J. Shavlik, "Transfer Learning," in *Handbook of Research on Machine Learning Applications*, Wisconsin, IGI Global, 2009.

[20] S. J. Pan and Q. Yang, "A Surevey on Transfer Learning," *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING,* 2009.

[21] R. Caruana, " Multitask learning," *Machine Learning,* vol. 28(1), pp. 41-75, 1997.

[22] S. Ruder, "Transfer Learning - Machine Learning's Next Frontier," 21 March 2017. [Online]. Available:

http://ruder.io/transfer-learning/. [Accessed 4 September 2018].

[23] "ImageNet," Stanford Vision Lab, Stanford University, 2016. [Online]. Available: http://image-net.org/index. [Accessed 2018].

[24] K. Simonyan and A. Zisserman, *VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION,* Visual Geometry Group, Department of Engineering Science, University of Oxford, 2015.

[25] "Keras," [Online]. Available: https://keras.io. [Online].

[26] "Global retail e-commerce market size 2014-2021," [Online]. Available: https://www.statista.com/. [Accessed 4 October 2018].

[27] "Natural Language Processing," [Online]. Available: https://www.sas.com. [Accessed 8 November 2018].

[28] A. Phophalia, "A Survey on Learning To Rank (LETOR) Approaches in Information Retrieval," Ahmedabad, 2011.

[29] A. Wall, "Search Engine History," 2006. [Online]. Available: http://www.searchenginehistory.com/. [Accessed 2 November 2018].

[30] T. Qin, T. Y. Liu, J. Xu and H. Li, "LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval".

[31] A. Prakash, "Techniques for Deep Query Understanding," ROORKEE.

[32] Z. Dou, R. Song and J.-R. Wen, "A Large-scale Evaluation and Analysis of Personalized Search Strategies," Beijing, 2007.

[33] Paul, "The Big Data Revolution Hits E-Commerce Search," [Online]. Available: https://www.searchtechnologies.com. [Accessed 16 October 2018].

[34] R. Bharadwaj, "NLP for eCommerce Search – Current Challenges and Future Potential," 22 July 2018. [Online]. Available: https://www.techemergence.com. [Accessed 2 November 2018].

[35] T. Fessenden, "Scrolling and Attention," 15 April 2018. [Online]. Available: https://www.nngroup.com. [Accessed 3 November 2018].

[36] D. Turnbull, "BM25 The Next Generation of Lucene Relevance," 16 October 2015. [Online]. Available: https://opensourceconnections.com. [Accessed 15 November 2018].

[37] "MOLE - Text Analysis Group," [Online]. Available: http://cogsys.imm.dtu.dk. [Accessed 4 November 2018].

[38] H. P. Luhn, "The Automatic Creation of Literature Abstracts," *IBM Journal of Research and Development,* vol. 2, no. 2, pp. 159 - 165 and 317, April 1958.

[39] G. Salton and C. Buckley, "Term Weighting Approaches in Automatic Text Retrieval," Information Processing and Management , 1987.

[40] G. Salton, "Introduction to Modern Information Retrieval," McGraw-Hill, 1983.

[41] H. Wang, H. Xiaodong, M.-W. Chang, S. Yang, R. W. White and W. Chu, "Personalized Ranking Model Adaptation for Web Search," Urbana.

[42] D. Tunkelang, "Query Understanding," 2016. [Online]. Available: https://queryunderstanding.com. [Accessed 7 October 2018].

[43] Y. Freund, R. Iyer, R. E. Schapire and Y. Singer, " An Efficient Boosting algorithm for combining preferences," *Journal of Machine Learning Research(JMLR),* vol. 4, pp. 933-969, 2003.

[44] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton and G. Hullender, "Learning to Rank using Gradient Descent," in *ICML*, 2002.

[45] A. Shashua and A. Levin, "Taxonomy of large margin principle algorithms".

[46] P. Li, C. Burges and Q. Wu, "McRank: Learning to Rank using multiple classification and gradient boosting," in *NIPS*, 2007.

[47] Z. Cao, T. Qin, T.-Y. Lin, M.-F. Tsai and H. Li, "Learning to Rank: From Pairwise Approach to Listwise Approach," in *ICML*, 2007.

[48] R. Deveaud, J. Mothe and M. Z. Ullah, "Learning to Adaptively Rank Document Retrieval System Configurations," IRIT.

[49] D. Britz, "Exploration vs Exploitation," April 2014. [Online]. Available: https://medium.com/@dennybritz. [Accessed 22 October 2018].

APPENDIX

Phase 1 – Comparison of transfer learning with CNN on subset of data


```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Using the traditional method by building a neural network from scratch"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "scrolled": true
   },
   "outputs": [],
   "source": [
    "import keras\n",
    "from keras.preprocessing.image import ImageDataGenerator, img_to_array, array_to_img, load_img\n",
    "from keras.models import Sequential\n",
    "from keras.layers import Conv2D, MaxPooling2D\n",
    "from keras.layers import Activation, Flatten, Dropout, Dense\n",
    "from keras import backend as K"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "import time"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "img_width, img_height = 200, 200\n",
    "train_data_dir = (\"E:\\\\Rashmeet_SJSU\\\\Sem IV\\\\CS 298\\\\subset_data\\\\train\")\n",
    "validation_data_dir = (\"E:\\\\Rashmeet_SJSU\\\\Sem IV\\\\CS 298\\\\subset_data\\\\validation\")\n",
    "nb_train_samples = 3039\n",
    "nb_validation_samples = 600\n",
    "epochs = 50\n",
    "batch_size = 16"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "K.image_data_format()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "start_time = time.time()\n",
    "if K.image_data_format() == 'channels_first':\n",
    "    input_shape = (3, img_width, img_height)\n",
    "else:\n",
    "    input_shape = (img_width, img_height, 3)\n",
    "print(input_shape)\n",
    "print(\"time taken = \", time.time() - start_time)\n",
    "\n",
    "#Input shape represents widht, height, and channels\n",
    "#Here channels value is 3 cuz of R, G, B"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "model = Sequential()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "#(3,3) is the filter -> since the input image is of depth 3 i.e. RGB, the filter will also be 3x3\n",
    "#32 represents the number of filters applied for depth\n",
    "#Filter takes input the 3D image and gives output corresponding 2D image for the same - https://www.youtube.com/watch?v=m8pOnJxOcqY&t=7s\n",
    "\n",
    "model.add(Conv2D(32, (3,3), input_shape = input_shape))\n",
    "model.add(Activation('relu'))\n",
    "\n",
    "model.add(MaxPooling2D(pool_size=(2,2)))\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "model.add(Conv2D(32,(3,3)))\n",
    "model.add(Activation('relu'))\n",
    "\n",
    "model.add(MaxPooling2D(pool_size=(2,2)))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "model.add(Conv2D(64,(3,3)))\n",
    "model.add(Activation('relu'))\n",
    "\n",
    "model.add(MaxPooling2D(pool_size=(2,2)))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "model.add(Flatten())\n",
    "model.add(Dense(64))\n",
    "model.add(Activation('relu'))\n",
    "model.add(Dropout(0.5))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "model.add(Dense(1))\n",
    "model.add(Activation('sigmoid'))\n"
   ]
  },
```

```
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"model.compile(loss='binary_crossentropy'
,\n",
   "          optimizer='rmsprop',\n",
   "          metrics=['accuracy'])"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "train_datagen =
ImageDataGenerator(rescale=1/255,\n",
   "                          shear_range=0.2,\n",
   "                          zoom_range=0.2,\n",
   "
horizontal_flip=True\n",
   "                          )"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "test_datagen=
ImageDataGenerator(rescale=1/255)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "train_generator =
train_datagen.flow_from_directory(train_d
ata_dir,\n",
   "                                    target_size
= (img_width, img_height),\n",
   "
batch_size= batch_size,\n",
   "
class_mode = 'binary'\n",
   "                                    )"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "validation_generator =
test_datagen.flow_from_directory(validati
on_data_dir,\n",
   "
target_size = (img_width, img_height),\n",
   "
batch_size = batch_size,\n",
   "
class_mode = 'binary')"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "start_time = time.time()\n",
   "history_traditional_cnn =
model.fit_generator(train_generator,\n",
   "                    steps_per_epoch =
nb_train_samples // batch_size,\n",
   "                    epochs = epochs,\n",
   "                    validation_data =
validation_generator,\n",
   "                    validation_steps = 4\n",
   "                   )\n",
   "\n",
   "print(\"Time taken by traditional CNN
process = \", time.time()-start_time)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "import matplotlib.pyplot as plt\n",
   "%matplotlib inline"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "import numpy as np"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "# Using transfer learning with a pre-
trained VGG16 model"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "model_imgnet =
applications.VGG16(include_top=False,
weights='imagenet')"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "train_datagen =
ImageDataGenerator(rescale=1./255,\n",
   "
zoom_range=0.3,\n",
   "
rotation_range=50,\n",
   "
width_shift_range=0.2,\n",
   "
height_shift_range=0.2,\n",
   "                          shear_range=0.2,
\n",
   "
horizontal_flip=True,\n",
   "
fill_mode='nearest'\n",
   "                          )"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "test_datagen=
ImageDataGenerator(rescale=1. /255)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "train_generator =
train_datagen.flow_from_directory(train_d
ata_dir,\n",
```

```
   "                                        target_size
= (img_width, img_height),\n",
   "
batch_size= batch_size,\n",
   "
class_mode = 'binary'\n",
   "                                       )"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "validation_generator =
test_datagen.flow_from_directory(validati
on_data_dir,\n",
   "
target_size = (img_width, img_height),\n",
   "
batch_size = batch_size,\n",
   "
class_mode = 'binary')"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "bottleneck_features_train =
model_imgnet.predict_generator(\n",
   "    train_generator,\n",
   "    nb_train_samples // batch_size\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\subset_data\\\\subset_bottleneck_
features_train.npy', 'wb'),\n",
   "       bottleneck_features_train)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "bottleneck_features_validation =
model_imgnet.predict_generator(\n",
   "    validation_generator,\n",
   "    nb_validation_samples //
batch_size\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\subset_data\\\\subset_bottleneck_
features_test.npy', 'wb'),\n",
   "       bottleneck_features_validation)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "train_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\subset_data\\\\subset_bottleneck_
features_train.npy',\n",
   "                      'rb'\n",
   "                      ))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "train_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "train_labels = np.array([0] * 2130  + [1]
* 894)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "validation_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\subset_data\\\\subset_bottleneck_
features_test.npy',\n",
   "                      'rb'\n",
   "                      ))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "validation_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "validation_labels = np.array([0]*492 +
[1]*100)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "from keras import optimizers"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "# model = Sequential()\n",
   "#
model.add(Flatten(input_shape=train_data
.shape[1:]))\n",
   "# model.add(Dense(256,
activation='relu'))\n",
   "# model.add(Dropout(0.5))\n",
   "# model.add(Dense(128,
activation='relu',
kernel_regularizer=regularizers.l2(0.0001)))
\n",
   "# model.add(Dense(1,
activation='sigmoid'))\n",
   "\n",
   "\n",
   "\n",
   "model = Sequential()\n",

"model.add(Flatten(input_shape=train_dat
a.shape[1:]))\n",
```

```
   "model.add(Dense(128,
activation='relu'))\n",
   "model.add(Dropout(0.5))\n",
   "model.add(Dense(1,
activation='sigmoid'))\n",
   "\n",
   "# model = Sequential()\n",
   "#
model.add(Flatten(input_shape=train_data
.shape[1:]))\n",
   "# model.add(Dense(256,
activation='relu', input_dim=input_shape,
kernel_regularizer=regularizers.l2(0.0001)))
\n",
   "# model.add(Dropout(0.5))\n",
   "# model.add(Dense(128,
activation='relu'))\n",
   "# model.add(Dropout(0.5))\n",
   "# model.add(Dense(1,
activation='sigmoid'))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "model.compile(optimizer= Adam(lr =
0.0003),\n",
   "            loss='binary_crossentropy',\n",
   "            metrics=['accuracy'])\n",
   "\n",
   "# model.compile(optimizer=
optimizers.RMSprop(),\n",
   "#
loss='binary_crossentropy',\n",
   "#            metrics=['accuracy'])\n",
   "\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "start_time = time.time()\n",
   "history_transfer_learning =
model.fit(train_data, train_labels,\n",
   "        epochs=epochs,\n",
   "        batch_size=batch_size,\n",
   "        validation_data=(validation_data,
validation_labels))\n",
   "print(\"Time taken by transfer learning =
\", time.time() - start_time)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"model.save_weights('bottleneck_subset_
model.h5')"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "\n",
   "f, (ax1, ax2) = plt.subplots(1, 2,
figsize=(12, 4))\n",
   "t = f.suptitle('Basic CNN Performance',
fontsize=12)\n",
   "f.subplots_adjust(top=0.85,
wspace=0.3)\n",
   "\n",
   "epoch_list = list(range(0,epochs))\n",
   "ax1.plot(epoch_list,
history_transfer_learning.history['acc'],
label='Train Accuracy')\n",
   "ax1.plot(epoch_list,
history_transfer_learning.history['val_acc'],
label='Validation Accuracy')\n",
   "ax1.set_xticks(np.arange(0, epochs,
5))\n",
   "ax1.set_ylabel('Accuracy Value')\n",
   "ax1.set_xlabel('Epoch')\n",
   "ax1.set_title('Accuracy')\n",
   "l1 = ax1.legend(loc=\"best\")\n",
   "\n",
   "ax2.plot(epoch_list,
history_transfer_learning.history['loss'],
label='Train Loss')\n",
   "ax2.plot(epoch_list,
history_transfer_learning.history['val_loss']
, label='Validation Loss')\n",
   "ax2.set_xticks(np.arange(0, epochs,
5))\n",
   "ax2.set_ylabel('Loss Value')\n",
   "ax2.set_xlabel('Epoch')\n",
   "ax2.set_title('Loss')\n",
   "l2 = ax2.legend(loc=\"best\")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "print(\"Average accuracy for transfer
learning is : \",
np.mean(history_transfer_learning.history[
'acc']))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "scrolled": false
  },
  "outputs": [],
  "source": [
   "\n",
   "f, (ax1, ax2) = plt.subplots(1, 2,
figsize=(12, 4))\n",
   "t = f.suptitle('Transfer Learning
Performance', fontsize=12)\n",
   "f.subplots_adjust(top=0.85,
wspace=0.3)\n",
   "\n",
   "epoch_list = list(range(0,epochs))\n",
   "ax1.plot(epoch_list,
history_traditional_cnn.history['acc'],
label='Train Accuracy')\n",
   "ax1.plot(epoch_list,
history_traditional_cnn.history['val_acc'],
label='Validation Accuracy')\n",
   "ax1.set_xticks(np.arange(0, epochs,
5))\n",
   "ax1.set_ylabel('Accuracy Value')\n",
   "ax1.set_xlabel('Epoch')\n",
   "ax1.set_title('Accuracy')\n",
   "l1 = ax1.legend(loc=\"best\")\n",
   "\n",
   "ax2.plot(epoch_list,
history_traditional_cnn.history['loss'],
label='Train Loss')\n",
   "ax2.plot(epoch_list,
history_traditional_cnn.history['val_loss'],
label='Validation Loss')\n",
   "ax2.set_xticks(np.arange(0, epochs,
5))\n",
   "ax2.set_ylabel('Loss Value')\n",
   "ax2.set_xlabel('Epoch')\n",
   "ax2.set_title('Loss')\n",
   "l2 = ax2.legend(loc=\"best\")"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "The above graphs show that the model
starts overfitting on the training data"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "print(\"Average accuracy for traditonal
CNN is : \",
np.mean(history_traditional_cnn.history['a
cc']))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "\n",

"print(bottleneck_feature_example.shape)
\n",
```

```
"plt.imshow(bottleneck_feature_example[
0][:,:,0])"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Test on appliance and furniture
images"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "from IPython.display import Image"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [

"Image(filename=\"E:\\\\\Rashmeet_SJSU\\
\\Sem IV\\\\CS
298\\\\data\\\\appliances\\\\07 (1).jpg\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [

"Image(filename='E:\\\\Rashmeet_SJSU\\\\
Sem IV\\\\CS
298\\\\data\\\\furniture\\\\19.jpg')"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "def load_image(infilename):\n",
    "    from PIL import Image\n",
    "    img = Image.open(infilename)\n",
    "    img.load()\n",
    "    data =
np.asarray(img,dtype=\"int32\")\n",
    "    return data"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "test_img =
load_image(\"E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\appliances\\\\07
(1).jpg\")\n",
    "test_img1 =
load_image('E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS
298\\\\data\\\\furniture\\\\19.jpg')"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "test_img.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "test_img1.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "from keras.preprocessing import image"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#resize appliance image\n",
    "img_original =
image.array_to_img(test_img,
scale=False)\n",
    "desired_width, desired_height = 200,
200\n",
    "width, height = 124, 200\n",
    "#start_x = np.maximum(0, int((width-
desired_width)/2))\n",
    "#img = img_original.crop((start_x,
np.maximum(0, -1), start_x+desired_width,
height))\n",
    "img = img_original.resize((200, 200))\n",
    "img_arr =
image.img_to_array(img)/255\n",
    "img_arr.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#resize furniture image\n",
    "img_original1 =
image.array_to_img(test_img1,
scale=False)\n",
    "desired_width, desired_height = 200,
200\n",
    "width, height = 124, 200\n",
    "#start_x = np.maximum(0, int((width-
desired_width)/2))\n",
    "#img = img_original.crop((start_x,
np.maximum(0, -1), start_x+desired_width,
height))\n",
    "img1 = img_original1.resize((200,
200))\n",
    "img_arr1 =
image.img_to_array(img1)/255\n",
    "img_arr1.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "features =
model_imgnet.predict(img_arr.reshape(1,
desired_width, desired_height, 3))\n",
    "features.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "features1 =
model_imgnet.predict(img_arr1.reshape(1,
desired_width, desired_height, 3))\n",
    "features1.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "predictions = model.predict(features)"
   ]
```

```
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions[0][0] < 0.5"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.predict_classes(features)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "predictions1 =
model.predict(features1)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions1[0][0] < 0.5"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.predict_classes(features1)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "train_generator.class_indices"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": []
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": []
 }
],
"metadata": {
 "kernelspec": {
  "display_name": "Python 3",
  "language": "python",
  "name": "python3"
 },
 "language_info": {
  "codemirror_mode": {
   "name": "ipython",
   "version": 3
  },
  "file_extension": ".py",
  "mimetype": "text/x-python",
  "name": "python",
  "nbconvert_exporter": "python",
  "pygments_lexer": "ipython3",
  "version": "3.6.3"
 }
},
"nbformat": 4,
"nbformat_minor": 2
}
```

Phase 2 – Application of transfer learning on ecommerce product images

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Load VGG16 Model and get features
for all classes together"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "from keras import applications\n",
    "from keras.models import
Sequential\n",
    "from keras.layers import Conv2D,
MaxPooling2D\n",
    "from keras.layers import Activation,
Dropout, Flatten, Dense\n",
    "from keras.preprocessing.image import
ImageDataGenerator, array_to_img,
img_to_array, load_img\n",
    "from keras import regularizers\n",
    "\n",
    "import numpy as np "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "datagen =
ImageDataGenerator(rescale=1. / 255,\n",
    "                    validation_split =
0.10)\n",
    "# test_datagen =
ImageDataGenerator(rescale= 1. / 255)\n",
    "batch_size = 16\n",
    "img_width = 200\n",
    "img_height = 200\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "model_imgnet =
applications.VGG16(include_top=False,
weights='imagenet')\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "train_generator =
datagen.flow_from_directory(\n",
    "    'E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS 298\\\\data\\\\all',\n",
    "    target_size = (img_width,
img_height),\n",
    "    batch_size= batch_size,\n",
    "    subset = 'training'\n",
    ")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "nb_train_samples = 8201\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "start_time = time.time()\n",
    "\n",
    "bottleneck_features_train =
model_imgnet.predict_generator(\n",
    "    train_generator,\n",
    "    nb_train_samples // batch_size\n",
    ")\n",
    "\n",
    "print(\"Time taken to generate
bottleneck features = \", time.time() -
start_time)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\all\\\\bottleneck_features_
train.npy', 'wb'),\n",
    "       bottleneck_features_train)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "validation_generator =
datagen.flow_from_directory(\n",
    "    'E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS 298\\\\data\\\\all',\n",
    "    target_size = (img_width,
img_height),\n",
    "    batch_size= batch_size,\n",
    "    subset = 'validation'\n",
    ")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "nb_validation_samples = 909"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "%timeit\n",
    "\n",
    "bottleneck_features_validation =
model_imgnet.predict_generator(\n",
    "    train_generator,\n",
    "    nb_validation_samples //
batch_size\n",
    ")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\all\\\\bottleneck_features_
test.npy', 'wb'),\n",
```

```
   "       bottleneck_features_validation)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "train_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Sem IV\\\\CS
298\\\\data\\\\all\\\\bottleneck_features_train.npy', 'rb'))\n",
   "validation_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Sem IV\\\\CS
298\\\\data\\\\all\\\\bottleneck_features_test.npy', 'rb'))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "len(train_data)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "len(validation_data)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "train_generator.class_indices"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "train_labels = np.array([0] * 1698  + [1] * 2948 + [2]*1546 + [3]*2000)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "validation_labels = np.array([0] * 188  + [1] * 320 + [2]*169 + [3]*219)"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "## Add the fully connected layer and fit
"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "model = Sequential()\n",

"model.add(Flatten(input_shape=train_data.shape[1:]))\n",
   "model.add(Dense(256,
activation='relu'))\n",
   "model.add(Dropout(0.5))\n",
   "model.add(Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(0.001)))\n",
   "model.add(Dense(1,
activation='sigmoid'))\n",
   "\n",
   "\n",
   "# model = Sequential()\n",
   "#
model.add(Flatten(input_shape=train_data.shape[1:]))\n",
   "# model.add(Dense(256,
activation='relu'))\n",
   "# model.add(Dropout(0.5))\n",
   "# model.add(Dense(1,
activation='sigmoid'))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "model.compile(optimizer='adam',\n",
   "              loss='binary_crossentropy',\n",
   "              metrics=['accuracy'])\n",
   "\n",
   "#
model.compile(optimizer='rmsprop',\n",
   "#
loss='binary_crossentropy',\n",
   "#              metrics=['accuracy'])"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "history = model.fit(train_data,
train_labels,\n",
   "          epochs= 50,\n",
   "          batch_size=batch_size,\n",
   "          validation_data=(validation_data,
validation_labels))"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "# For each model individually"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "appliances_train_generator =
datagen.flow_from_directory(\n",
   "    'E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS
298\\\\data\\\\appliances_images',\n",
   "    target_size = (img_width,
img_height),\n",
   "    batch_size= batch_size,\n",
   "    subset = 'training'\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "nb_appliances_train_samples = 1698"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "appliances_bottleneck_features_train =
model_imgnet.predict_generator(\n",
   "    appliances_train_generator,\n",
   "    nb_appliances_train_samples //
batch_size\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
```

```
   "collapsed": true
  },
  "outputs": [],
  "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\appliances_bottleneck_feat
ures_train.npy', 'wb'),\n",
   "
appliances_bottleneck_features_train)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "appliances_validation_generator =
datagen.flow_from_directory(\n",
   "    'E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS
298\\\\data\\\\appliances_images',\n",
   "    target_size = (img_width,
img_height),\n",
   "    batch_size= batch_size,\n",
   "    subset = 'validation'\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "nb_appliances_validation_samples =
188"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"appliances_bottleneck_features_validatio
n = model_imgnet.predict_generator(\n",
   "    appliances_validation_generator,\n",
   "    nb_appliances_validation_samples //
batch_size\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\appliances_bottleneck_feat
ures_test.npy', 'wb'),\n",
   "
appliances_bottleneck_features_validation
)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "electronics_train_generator =
datagen.flow_from_directory(\n",
   "    'E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS
298\\\\data\\\\electronics_images',\n",
   "    target_size = (img_width,
img_height),\n",
   "    batch_size= batch_size,\n",
   "    subset = 'training'\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "nb_electronics_train_samples = 2951"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "electronics_bottleneck_features_train =
model_imgnet.predict_generator(\n",
   "    electronics_train_generator,\n",
   "    nb_electronics_train_samples //
batch_size\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\electronics_bottleneck_fea
tures_train.npy', 'wb'),\n",
   "
electronics_bottleneck_features_train)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "electronics_validation_generator =
datagen.flow_from_directory(\n",
   "    'E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS
298\\\\data\\\\electronics_images',\n",
   "    target_size = (img_width,
img_height),\n",
   "    batch_size= batch_size,\n",
   "    subset = 'validation'\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "nb_electronics_validation_samples =
327"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"electronics_bottleneck_features_validatio
n = model_imgnet.predict_generator(\n",
   "    electronics_validation_generator,\n",
   "    nb_electronics_validation_samples //
batch_size\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\electronics_bottleneck_fea
tures_test.npy', 'wb'),\n",
   "
electronics_bottleneck_features_validation
)"
  ]
```

```
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "furniture_train_generator =
datagen.flow_from_directory(\n",
   "    'E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS
298\\\\data\\\\furniture_images',\n",
   "    target_size = (img_width,
img_height),\n",
   "    batch_size= batch_size,\n",
   "    subset = 'training'\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "nb_furniture_train_samples = 1548"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "furniture_bottleneck_features_train =
model_imgnet.predict_generator(\n",
   "    furniture_train_generator,\n",
   "    nb_furniture_train_samples //
batch_size\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\furniture_bottleneck_featu
res_train.npy', 'wb'),\n",
   "
furniture_bottleneck_features_train)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "furniture_validation_generator =
datagen.flow_from_directory(\n",
   "    'E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS
298\\\\data\\\\furniture_images',\n",
   "    target_size = (img_width,
img_height),\n",
   "    batch_size= batch_size,\n",
   "    subset = 'validation'\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "nb_furniture_validation_samples = 172"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "furniture_bottleneck_features_test =
model_imgnet.predict_generator(\n",
   "    furniture_validation_generator,\n",
   "    nb_furniture_validation_samples //
batch_size\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\furniture_bottleneck_featu
res_test.npy', 'wb'),\n",
   "
furniture_bottleneck_features_test)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "toys_train_generator =
datagen.flow_from_directory(\n",
   "    'E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS 298\\\\data\\\\toys_images',\n",
   "    target_size = (img_width,
img_height),\n",
   "    batch_size= batch_size,\n",
   "    subset = 'training'\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "nb_toys_train_samples = 2004"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "toys_bottleneck_features_train =
model_imgnet.predict_generator(\n",
   "    toys_train_generator,\n",
   "    nb_toys_train_samples //
batch_size\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\toys_bottleneck_features_t
rain.npy', 'wb'),\n",
   "      toys_bottleneck_features_train)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "toys_validation_generator =
datagen.flow_from_directory(\n",
   "    'E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS 298\\\\data\\\\toys_images',\n",
   "    target_size = (img_width,
img_height),\n",
   "    batch_size= batch_size,\n",
   "    subset = 'validation'\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
```

```
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "nb_toys_validation_samples = 222"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "toys_bottleneck_features_test =
model_imgnet.predict_generator(\n",
   "    toys_validation_generator,\n",
   "    nb_toys_validation_samples //
batch_size\n",
   ")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [

"np.save(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\toys_bottleneck_features_t
est.npy', 'wb'),\n",
   "      toys_bottleneck_features_test)"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "# Load Features"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "appliances_train_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\appliances_bottleneck_feat
ures_train.npy', 'rb'))\n",
   "electronics_train_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\electronics_bottleneck_fea
tures_train.npy', 'rb'))\n",
   "furniture_train_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\furniture_bottleneck_featu
res_train.npy', 'rb'))\n",
   "toys_train_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\toys_bottleneck_features_t
rain.npy', 'rb'))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "appliances_train_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "electronics_train_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "furniture_train_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "toys_train_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "appliances_validation_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\appliances_bottleneck_feat
ures_test.npy', 'rb'))\n",
   "electronics_validation_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\electronics_bottleneck_fea
tures_test.npy', 'rb'))\n",
   "furniture_validation_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\furniture_bottleneck_featu
res_test.npy', 'rb'))\n",
   "toys_validation_data =
np.load(open('E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\data\\\\toys_bottleneck_features_t
est.npy', 'rb'))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "appliances_validation_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "electronics_validation_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "furniture_validation_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "toys_validation_data.shape"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "## Train Appliances Model"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "from keras import optimizers"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
```

```
  "outputs": [],
  "source": [
   "appliances_train =
np.vstack((appliances_train_data[:],
electronics_train_data[:800],
furniture_train_data[:800],
toys_train_data[:800]))\n",
   "\n",
   "appliances_validation =
np.vstack((appliances_validation_data[:],
electronics_validation_data[:100],\n",
   "
furniture_validation_data[:100],
toys_validation_data[:100]))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "appliances_train_labels = np.array([0] *
1696 + [1] * 2400)\n",
   "appliances_validation_labels =
np.array([0] * 176 + [1] * 300)\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "batch_size = 16\n",
   "model = Sequential()\n",

"model.add(Flatten(input_shape=appliance
s_train.shape[1:]))\n",
   "model.add(Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(0.0001)))
\n",
   "model.add(Dropout(0.5))\n",
   "# model.add(Dense(128,
activation='relu',
kernel_regularizer=regularizers.l2(0.01)))\n
",
   "model.add(Dense(1,
activation='sigmoid'))\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "model.compile(optimizer=
optimizers.Adam(lr = 1e-5),\n",
   "          loss='binary_crossentropy',\n",
   "          metrics=['accuracy'])\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "scrolled": true
  },
  "outputs": [],
  "source": [
   "\n",
   "appliance_history =
model.fit(appliances_train,
appliances_train_labels,\n",
   "        epochs = 12,\n",
   "        batch_size = batch_size,\n",
   "
validation_data=(appliances_validation,
appliances_validation_labels))\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "model.save('appliances_model.h5')"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "import matplotlib.pyplot as plt\n",
   "%matplotlib inline"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "\n",
   "f, (ax1, ax2) = plt.subplots(1, 2,
figsize=(12, 4))\n",
   "t = f.suptitle('Appliances Model
Performance', fontsize=12)\n",
   "f.subplots_adjust(top=0.85,
wspace=0.3)\n",
   "\n",
   "epoch_list = list(range(0,12))\n",
   "ax1.plot(epoch_list,
appliance_history.history['acc'],
label='Train Accuracy')\n",
   "ax1.plot(epoch_list,
appliance_history.history['val_acc'],
label='Validation Accuracy')\n",
   "ax1.set_xticks(np.arange(0, 12, 5))\n",
   "ax1.set_ylabel('Accuracy Value')\n",
   "ax1.set_xlabel('Epoch')\n",
   "ax1.set_title('Accuracy')\n",
   "l1 = ax1.legend(loc=\"best\")\n",
   "\n",
   "ax2.plot(epoch_list,
appliance_history.history['loss'],
label='Train Loss')\n",
   "ax2.plot(epoch_list,
appliance_history.history['val_loss'],
label='Validation Loss')\n",
   "ax2.set_xticks(np.arange(0, 12, 5))\n",
   "ax2.set_ylabel('Loss Value')\n",
   "ax2.set_xlabel('Epoch')\n",
   "ax2.set_title('Loss')\n",
   "l2 = ax2.legend(loc=\"best\")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "print(\"Average accuracy for appliances
model is : \",
np.mean(appliance_history.history['acc']))"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "# Load all images and prepare features
to test"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "from IPython.display import Image"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [

"Image(filename=\"E:\\\\Rashmeet_SJSU\\
\\Sem IV\\\\CS
298\\\\test_imgs\\\\appliances2.jpg\")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [

"Image(filename=\"E:\\\\Rashmeet_SJSU\\
\\Sem IV\\\\CS
298\\\\test_imgs\\\\furniture1.jpg\")"
  ]
```

```
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [

"Image(filename=\"E:\\\\\\Rashmeet_SJSU\\\
\\Sem IV\\\\\CS
298\\\\\test_imgs\\\\\electronics1.jpg\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [

"Image(filename=\"E:\\\\\\Rashmeet_SJSU\\\
\\Sem IV\\\\\CS
298\\\\\test_imgs\\\\\toy1.jpg\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "def load_image(infilename):\n",
    "    from PIL import Image\n",
    "    img = Image.open(infilename)\n",
    "    img.load()\n",
    "    data =
np.asarray(img,dtype=\"int32\")\n",
    "    return data"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "test_img_app =
load_image(\"E:\\\\\\Rashmeet_SJSU\\\\\Se
m IV\\\\\CS
298\\\\\test_imgs\\\\\appliances2.jpg\")\n",
    "test_img_furn =
load_image(\"E:\\\\\\Rashmeet_SJSU\\\\\Se
m IV\\\\\CS
298\\\\\test_imgs\\\\\furniture2.jpg\")\n",
    "test_img_elec =
load_image(\"E:\\\\\\Rashmeet_SJSU\\\\\Se
m IV\\\\\CS
298\\\\\test_imgs\\\\\electronics1.jpg\")\n",
    "test_img_toys =
load_image(\"E:\\\\\\Rashmeet_SJSU\\\\\Se
m IV\\\\\CS
298\\\\\test_imgs\\\\\toy1.jpg\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "test_img_app.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "test_img_furn.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "test_img_elec.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "test_img_toys.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "from keras.preprocessing import image"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#resize appliance image\n",
    "img_app_orig =
image.array_to_img(test_img_app,
scale=False)\n",
    "desired_width, desired_height = 200,
200\n",
    "width, height = 124, 200\n",
    "#start_x = np.maximum(0, int((width-
desired_width)/2))\n",
    "#img = img_original.crop((start_x,
np.maximum(0, -1), start_x+desired_width,
height))\n",
    "img_app = img_app_orig.resize((200,
200))\n",
    "img_app_arr =
image.img_to_array(img_app)/255\n",
    "img_app_arr.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#resize furniture image\n",
    "img_furn_orig =
image.array_to_img(test_img_furn,
scale=False)\n",
    "desired_width, desired_height = 200,
200\n",
    "width, height = 124, 200\n",
    "#start_x = np.maximum(0, int((width-
desired_width)/2))\n",
    "#img = img_original.crop((start_x,
np.maximum(0, -1), start_x+desired_width,
height))\n",
    "img_furn = img_furn_orig.resize((200,
200))\n",
    "img_furn_arr =
image.img_to_array(img_furn)/255\n",
    "img_furn_arr.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#resize electronics image\n",
    "img_elec_orig =
image.array_to_img(test_img_elec,
scale=False)\n",
    "desired_width, desired_height = 200,
200\n",
    "width, height = 124, 200\n",
    "#start_x = np.maximum(0, int((width-
desired_width)/2))\n",
    "#img = img_original.crop((start_x,
np.maximum(0, -1), start_x+desired_width,
height))\n",
    "img_elec = img_elec_orig.resize((200,
200))\n",
    "img_elec_arr =
image.img_to_array(img_elec)/255\n",
    "img_elec_arr.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#resize toys image\n",
    "img_toys_orig =
image.array_to_img(test_img_toys,
scale=False)\n",
    "desired_width, desired_height = 200,
200\n",
    "width, height = 124, 200\n",
    "#start_x = np.maximum(0, int((width-
desired_width)/2))\n",
    "#img = img_original.crop((start_x,
```

```
np.maximum(0, -1), start_x+desired_width, height))\n",
   "img_toys = img_toys_orig.resize((200, 200))\n",
   "img_toys_arr = image.img_to_array(img_toys)/255\n",
   "img_toys_arr.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "features_app = model_imgnet.predict(img_app_arr.reshape(1, desired_width, desired_height, 3))\n",
   "features_app.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "features_furn = model_imgnet.predict(img_furn_arr.reshape(1, desired_width, desired_height, 3))\n",
   "features_furn.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "features_elec = model_imgnet.predict(img_elec_arr.reshape(1, desired_width, desired_height, 3))\n",
   "features_elec.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "features_toys = model_imgnet.predict(img_toys_arr.reshape(1, desired_width, desired_height, 3))\n",
   "features_toys.shape"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "# Test appliances model"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "plt.imshow(features_app[0][:,:,0])"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [

"appliances_train_generator.class_indices\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions_app = model.predict(features_app)\n",
   "print(predictions_app[0][0] < 0.5)\n",
   "model.predict_classes(features_app)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions_furn = model.predict(features_furn)\n",
   "print(predictions_furn[0][0] < 0.5)\n",
   "model.predict_classes(features_furn)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions_elec = model.predict(features_elec)\n",
   "print(predictions_elec[0][0] < 0.5)\n",
   "model.predict_classes(features_elec)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions_toys = model.predict(features_toys)\n",
   "print(predictions_toys[0][0] < 0.5)\n",
   "model.predict_classes(features_toys)"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "## Train electronics model"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "electronics_train = np.vstack((electronics_train_data[:], appliances_train_data[:800],furniture_train_data[:800], toys_train_data[:800]))\n",
   "\n",
   "electronics_validation = np.vstack((electronics_validation_data[:], appliances_validation_data[:100], \n",
   "
furniture_validation_data[:100], toys_validation_data[:100]))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "electronics_train_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "electronics_validation_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "electronics_train_labels = np.array([0] * 2944 + [1] * 2400)\n",
   "electronics_validation_labels = np.array([0] * 320 + [1] * 300)\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "batch_size = 16\n",
   "model = Sequential()\n",

"model.add(Flatten(input_shape=electroni
```

```
cs_train.shape[1:]))\n",
   "model.add(Dense(256,
activation='relu'))\n",
   "model.add(Dropout(0.5))\n",
   "model.add(Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(0.01)))\n
",
   "model.add(Dense(1,
activation='sigmoid'))\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "model.compile(optimizer='adam',\n",
   "            loss='binary_crossentropy',\n",
   "            metrics=['accuracy'])\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.fit(electronics_train,
electronics_train_labels,\n",
   "        epochs = 12,\n",
   "        batch_size = batch_size,\n",
   "
validation_data=(electronics_validation,
electronics_validation_labels))\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "model.save('electronics_model.h5')"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "# Test electronics model"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "from IPython.display import Image"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [

"Image(filename=\"E:\\\\Rashmeet_SJSU\\
\\Sem IV\\\\CS
298\\\\subset_data\\\\train\\\\appliances\
\\\appliances37.png\")"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [

"Image(filename='E:\\\\Rashmeet_SJSU\\\\
Sem IV\\\\CS
298\\\\subset_data\\\\train\\\\furniture\\
\\furniture24.png')"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [

"Image(filename='E:\\\\Rashmeet_SJSU\\\\
Sem IV\\\\CS
298\\\\data\\\\electronics_images\\\\elect
ronics\\\\301 (1).jpg')"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "def load_image(infilename):\n",
   "    from PIL import Image\n",
   "    img = Image.open(infilename)\n",
   "    img.load()\n",
   "    data =
np.asarray(img,dtype=\"int32\")\n",
   "    return data"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "test_img_appliance =
load_image(\"E:\\\\Rashmeet_SJSU\\\\Se
m IV\\\\CS
298\\\\subset_data\\\\train\\\\appliances\
\\\appliances37.png\")\n",
   "test_img_furniture =
load_image('E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS
298\\\\subset_data\\\\train\\\\furniture\\
\\furniture24.png')\n",
   "test_img_electronics =
load_image('E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS
298\\\\data\\\\electronics_images\\\\elect
ronics\\\\301 (1).jpg')\n",
   "test_img_toys =
load_image('E:\\\\Rashmeet_SJSU\\\\Sem
IV\\\\CS 298\\\\test_imgs\\\\toy1.jpg')"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "test_img_appliance.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "test_img_furniture.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "test_img_electronics.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "test_img_toys.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "from keras.preprocessing import image"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
```

```
   "source": [
    "#resize appliance image\n",
    "img_original =
image.array_to_img(test_img_appliance,
scale=False)\n",
    "desired_width, desired_height = 200,
200\n",
    "width, height = 124, 200\n",
    "#start_x = np.maximum(0, int((width-
desired_width)/2))\n",
    "#img = img_original.crop((start_x,
np.maximum(0, -1), start_x+desired_width,
height))\n",
    "img = img_original.resize((200, 200))\n",
    "img_arr =
image.img_to_array(img)/255\n",
    "img_arr.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#resize furniture image\n",
    "img_original1 =
image.array_to_img(test_img_furniture,
scale=False)\n",
    "desired_width, desired_height = 200,
200\n",
    "width, height = 124, 200\n",
    "#start_x = np.maximum(0, int((width-
desired_width)/2))\n",
    "#img = img_original.crop((start_x,
np.maximum(0, -1), start_x+desired_width,
height))\n",
    "img1 = img_original1.resize((200,
200))\n",
    "img_arr1 =
image.img_to_array(img1)/255\n",
    "img_arr1.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#resize furniture image\n",
    "img_original2 =
image.array_to_img(test_img_electronics,
scale=False)\n",
    "desired_width, desired_height = 200,
200\n",
    "width, height = 124, 200\n",
    "#start_x = np.maximum(0, int((width-
desired_width)/2))\n",
    "#img = img_original.crop((start_x,
np.maximum(0, -1), start_x+desired_width,
height))\n",
    "img2 = img_original2.resize((200,
200))\n",
    "img_arr2 =
image.img_to_array(img2)/255\n",
    "img_arr2.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#resize furniture image\n",
    "img_original3 =
image.array_to_img(test_img_toys,
scale=False)\n",
    "desired_width, desired_height = 200,
200\n",
    "width, height = 124, 200\n",
    "#start_x = np.maximum(0, int((width-
desired_width)/2))\n",
    "#img = img_original.crop((start_x,
np.maximum(0, -1), start_x+desired_width,
height))\n",
    "img3 = img_original3.resize((200,
200))\n",
    "img_arr3 =
image.img_to_array(img3)/255\n",
    "img_arr3.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "features =
model_imgnet.predict(img_arr.reshape(1,
desired_width, desired_height, 3))\n",
    "features.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "features1 =
model_imgnet.predict(img_arr1.reshape(1,
desired_width, desired_height, 3))\n",
    "features1.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "features2 =
model_imgnet.predict(img_arr2.reshape(1,
desired_width, desired_height, 3))\n",
    "features2.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "features3 =
model_imgnet.predict(img_arr3.reshape(1,
desired_width, desired_height, 3))\n",
    "features3.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "predictions = model.predict(features)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "predictions[0][0] < 0.5\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "model.predict_classes(features)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "predictions1 =
model.predict(features1)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "predictions1[0][0] < 0.5"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "model.predict_classes(features1)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
```

```
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "predictions2 =
model.predict(features2)"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions2[0][0] < 0.5"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.predict_classes(features2)"
  ]
  },
  {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "# Train furniture model"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "furniture_train =
np.vstack((furniture_train_data[:],
appliances_train_data[:800],electronics_tra
in_data[:800], toys_train_data[:800]))\n",
   "\n",
   "furniture_validation =
np.vstack((furniture_validation_data[:],
appliances_validation_data[:100], \n",
   "
electronics_validation_data[:100],
toys_validation_data[:100]))"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "furniture_train_data.shape"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "furniture_validation_data.shape"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "furniture_train_labels = np.array([0] *
1536 + [1] * 2400)\n",
   "furniture_validation_labels =
np.array([0] * 160 + [1] * 300)\n"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "batch_size = 16\n",
   "model = Sequential()\n",

"model.add(Flatten(input_shape=furniture
_train.shape[1:]))\n",
   "model.add(Dense(256,
activation='relu'))\n",
   "model.add(Dropout(0.5))\n",
   "model.add(Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(0.01)))\n
",
   "model.add(Dense(1,
activation='sigmoid'))\n"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "model.compile(optimizer='adam',\n",
   "          loss='binary_crossentropy',\n",
   "          metrics=['accuracy'])\n"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.fit(furniture_train,
furniture_train_labels,\n",
   "       epochs = 12,\n",
   "       batch_size = batch_size,\n",
   "
validation_data=(furniture_validation,
furniture_validation_labels))\n"
  ]
  },
  {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "# Test furniture model"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.predict_classes(features)"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "predictions1 =
model.predict(features1)"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions1[0][0] < 0.5"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.predict_classes(features1)"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "predictions2 =
model.predict(features2)"
  ]
  },
  {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions2[0][0] < 0.5"
  ]
```

```
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.predict_classes(features2)"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "# Train toys model"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "toys_train =
np.vstack((toys_train_data[:],
appliances_train_data[:800],electronics_tra
in_data[:800],
furniture_train_data[:800]))\n",
   "\n",
   "toys_validation =
np.vstack((toys_validation_data[:],
appliances_validation_data[:100], \n",
   "
electronics_validation_data[:100],
furniture_validation_data[:100]))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "toys_train_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "toys_validation_data.shape"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "toys_train_labels = np.array([0] * 2000 +
[1] * 2400)\n",
   "toys_validation_labels = np.array([0] *
208 + [1] * 300)\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "batch_size = 16\n",
   "model = Sequential()\n",

"model.add(Flatten(input_shape=toys_trai
n.shape[1:]))\n",
   "model.add(Dense(256,
activation='relu'))\n",
   "model.add(Dropout(0.5))\n",
   "model.add(Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(0.01)))\n
",
   "model.add(Dense(1,
activation='sigmoid'))\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "model.compile(optimizer='adam',\n",
   "          loss='binary_crossentropy',\n",
   "          metrics=['accuracy'])"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.fit(toys_train,
toys_train_labels,\n",
   "       epochs = 12,\n",
   "       batch_size = batch_size,\n",
   "       validation_data=(toys_validation,
toys_validation_labels))\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "predictions = model.predict(features)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions[0][0] < 0.5"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.predict_classes(features)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "predictions1 =
model.predict(features1)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions1[0][0] < 0.5"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "model.predict_classes(features1)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "predictions2 =
model.predict(features2)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "predictions2[0][0] < 0.5"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
```

```
   "metadata": {},
   "outputs": [],
   "source": [
    "model.predict_classes(features2)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "predictions3 =
model.predict(features3)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "predictions3[0][0] < 0.5"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "model.predict_classes(features3)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.6.3"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```