Machine Learning Engineer Nanodegree

Capstone Project:

Large-Scale Item Categorization in E-Commerce

Jin Lu
February 22, 2018

I. **Definition**

Project Overview

The advances in web and mobile technology in recent years significantly boost on-line shopping activities. Both major e-commerce giants like eBay and Amazon and other rising stars of e-commerce platforms witnessed a rapid booming of their product catalogs and items in the past few years. Those e-commerce service providers are striving nowadays to apply machine learning techniques in the automatic classification of their on-line products, since it is impossible to handle this overwhelmingly large and fast growing data set by human effort. A precise classification of the constantly emerging new products is crucial for improving the user experience of those e-commerce platforms, since it enables the users to browser intuitively through a large range of products, which can potentially satisfy their needs.

Currently, the automatic prediction of product categories of e-commence platforms like Cdiscount.com is mainly based on the text description of the products [1]. To handle the large number of product categories, sometimes a hierarchical classification approach is also employed [2]. However, such metadata-based techniques seem now close to reach their maximum potential [3]. It is believed by some e-commerce platform providers that the next leap in the accuracy of automated product categorization relies on computer vision techniques. Therefore, Cdiscount - a major on-line retailer in France hosted a Kaggle competition by the end of 2017, namely the "Cdiscount's Image Classification Challenge", which aimed at making improved classification by using the images of the products provided by the users.

Problem Statement

In this project, the large-scale item categorization problem raised in the Kaggle competition "Cdiscount's Image Classification Challenge" will be addressed. The aim is to train a successful image-based product classifier on the labeled training sets provided by Cdiscount, which consists of over 10 million of real product images that belong to thousands of categories. A typical solution to such an image classification problem is to train convolutional neural network (CNN) with a well-tested architecture like VGG. However, this project is especially challenging due to the large size and high density of the semantic space that arise from large number of categories [4]. In addition, the high category skewness and the large size of the training data can pose further challenges in training a successful model.

To successfully handle the challenges of this large item categorization problem, a solution consisting of two consecutive classification steps is proposed. In the first step, a primitive prediction will be made among all the 5270 categories by a CNN model, which applies a well-established architecture for image classification (ResNet50). In the training of such a model, transfer learning will be applied, which utilizes the pre-trained weights on the 'ImageNet' datasets. Meanwhile, the original fully-connected layers at the top of pre-trained network will be replaced by a new fully-connected layer. The number of neurons in this layer is same as the number of categories. Firstly, only the weights of this newly added layer will be trained. All the pre-trained weights of other layers will remain the same. Then, in the second fine-tuning step, all the layers of the network will be set as trainable. The whole model will be retrained on the training set with a much reduced learning rate.

In the second step, according to the predicted label by the previous CNN model, the image will be passed to a corresponding XGBoost classifier. This second classifier takes the bottleneck features of the image from the CNN model as the input and is aimed at discriminating a small set of categories, which tend to be confused with the previously predicted category. Therefore in the training of this second classifier, a reduced train set will be used, which contains images only from the predicted category and from related highly confusable categories. It is expected that through such two consecutive classification steps high prediction accuracy can be achieved even for problems with very huge number of categories.

Metrics

The quality of the model will be evaluated by the prediction accuracy and the F1 score on the test set, the size of which is about 1/8 of the training set. The test set contains samples from all the 5270 categories, which are distributed as balanced as possible. It will be kept aside from the training set from the beginning. The prediction accuracy is calculated as the ratio between the number of correctly predicted labels and the total number of the samples in the test set.

$$\text{accuracy} = \frac{\text{number of correctly categorized images}}{\text{total number of images in the training set}} \qquad (1)$$

The F1 score is calculated based on the precision and recall on the test set:

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \qquad (2)$$

where $\text{precision} = \frac{\text{true positives}}{\text{true positives+false positives}}$, $\text{recall} = \frac{\text{true positives}}{\text{true positives+false negatives}}$.

II. Analysis

Data Exploration

The datasets are provided by the company Cdiscount, which can be downloaded from the Kaggle website [3]. These datasets are highly relevant to the problem described in the previous section, since according to Cdiscount, the products that appear in the datasets actually cover half of their present catalog and thus rather representative. A sample of the datasets is presented in Figure 1.

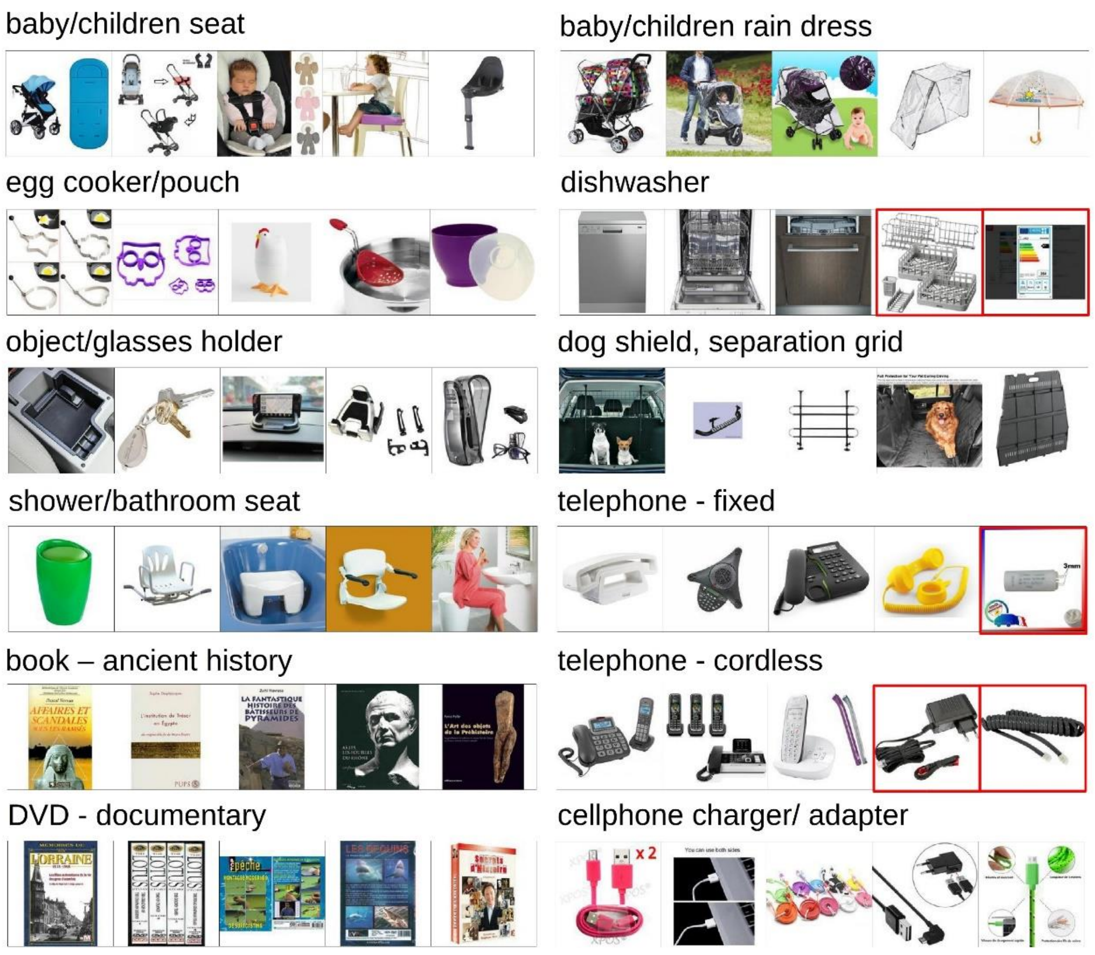


Figure 1. Representative training images from 12 randomly chosen categories. The images marked by red frames are considered as noisy training images, which display partial components of the product, relevant technical drawings or simply objects that are misclassified.

The file of the training set “train.bson” downloaded from the Kaggle website contains 15 million images (resolution: 180 x 180) belonging to 9 million products that come from the on-line shops of Cdiscount. The products cover a broad spectrum of everyday objects like clothes, cosmetics, CD/books, toys, electronic devices, auto accessories and so on. The images from those products are labeled as 5270 different categories.

It can be seen from Figure 1 that those real-life images contained in the training set show extremely large diversity. For example, the products that belong to the same category can be in rather great variety of forms. For the category of baby/children seat, depending on whether the products will be mounted in a baby carriage, in a car, on a chair or maybe on a bike, they can have very distinct exterior appearances. The products of the category egg cooker/pouch also have many various designs and shapes. In additions, the products are displayed in quite different styles in the images. The simplest style is the solo presentation of the product itself. However, they can also be shown in a complex scene related with their applications, presented in a series of different configurations, or marked with descriptive annotations. Such a great diversity of the images would increase the difficulty in training the model. Therefore, the model needs to generalize rather well to handle such real-life complexity.

The training data is also considered to be noisy and rather confusable in some parts. As marked by red squares in Figure 1, occasionally misclassified images can be found in the dataset. Sometimes, the images don't display the whole products but their components, partial views or even technical drawings/specifications, based on which it would be rather hard to make a correct classification. In addition, some categories are rather close in semantic space. For example, the categories of fixed telephone and cordless telephone shown in Figure 1. Another example is that the training set contains 162 categories of books like history, philosophy and politics books and 16 categories of DVDs like films, educational programs and music. Presumably, they would be highly confusable if the texts on the covers of books or DVDs are not extracted.

Exploratory Visualization

Among different categories, the distribution of their instances is highly imbalanced. In Figure 2 the number of images for each category is plotted in ascending order. The rarest category contains only 12 images whereas the most abundant category contains 80348 images. As it can be seen, roughly half of the categories contain less than 360 images. Therefore, in my training set, I decide to include maximum 300 images for each category. The setting of such a limit helps to balance the distribution of different categories in the training set and to prevent spending too much computational power on a few categories, which are extraordinarily abundant.

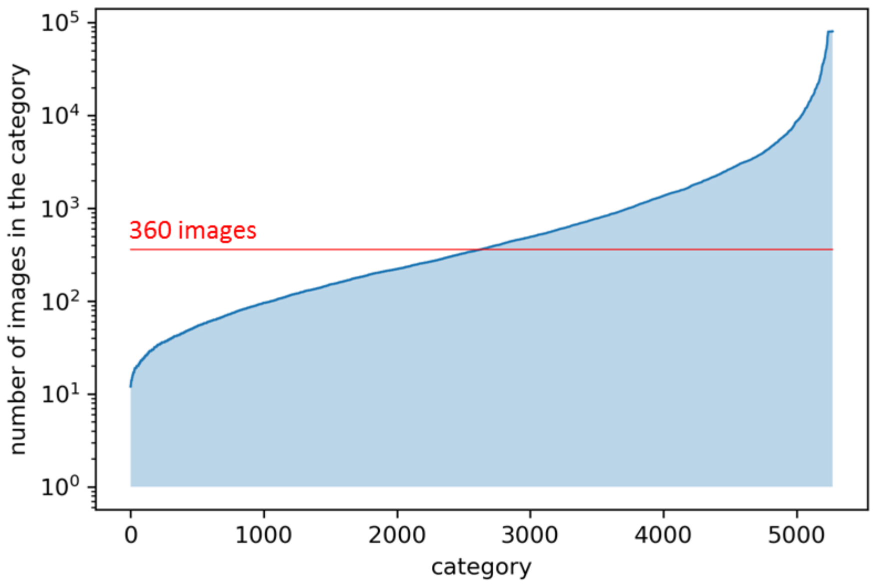


Figure 2. The number of images across 5270 categories in ascending order.

Algorithms and Techniques

The machine learning algorithms and techniques applied in this project are residual deep learning, transfer learning and XGBoosting.

Image classification problems as in this project are nowadays commonly handled by convolutional deep neural networks (previously referred as CNN). This technique is very capable in extracting abstract and sophisticated high-level concepts like the categories of objects present in the images simply based on the pixel arrays of the images in very complex real-life environment.

The success of this technique lies in that, in analogy to the cognition of human-beings, where our understanding about complex high-level concepts is achieved through their relations to simpler and lower-level concepts, deep neural networks can understand the world in such a nested hierarchy of concepts through their multiple hidden layers. In their feed-forward information flow in processing images, the early layers close to the input are usually dealing with the simplest features like edges. Then, the following layers can further detect corners and contours based on those simple features from the previous layers. Finally, the last few layers close to the output can respond with even higher-level features like object parts based on the previously outputted middle-level features.

The convolutional deep neural networks are especially suitable for processing data with grid-like topology such as images, which are actually 2-D arrays of pixel values. Because the convolution operation, which is based on the weight-sharing architecture, enables the neural network to successfully detect features irrespective of their locations in the image. This characteristic greatly simplifies the feature detection processes compared to the fully connected artificial neural network.

The power of achieving high-level abstraction of deep neural networks has its root in the multiple hidden layers. However, as more layers are added to the network, it becomes harder to get trained. The training accuracy firstly become saturated and then starts to decrease. This problem is also called degradation. To conquer this problem, a technique called deep residual learning is proposed. The major differences between conventional deep learning and the residual deep learning [5] are shown in Figure 3.

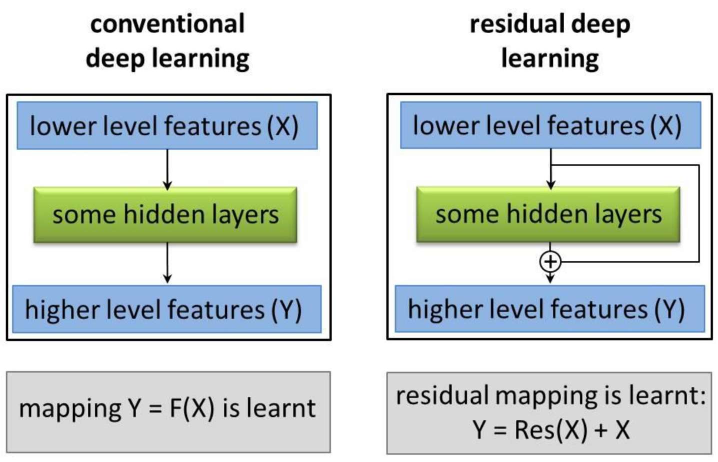

conventional deep learning
lower level features (X)
some hidden layers
higher level features (Y)
mapping Y = F(X) is learnt
residual deep learning
lower level features (X)
some hidden layers
higher level features (Y)
residual mapping is learnt: Y = Res(X) + X

Figure 3. The differences between the conventional deep learning and the residual deep learning [5].

As shown in the Figure above, for conventional deep learning, the building blocks of the deep neural network, namely the hidden layers of perceptions, are considered to learn certain mappings from lower level features to higher level features during training. However, in the residual deep learning, the input of the hidden layers will be added to the original output before they are fed as the input to the subsequent layers. Therefore, the hidden layers in the residual learning actually learn the difference (the residuals) between the input and the desired output. Due to these shortcut paths that connect both inputs and outputs of the hidden layers, the gradients propagate backwards much quicker in the training of the deep neural networks, which make it possible to train very deep neural networks. In addition, the training error of a very deep residual network is expected to be no greater than its shallower counterpart since the solver should be able to approximate the identical mappings achieved by a shallower network by setting the weights of certain hidden layers to zero. For those layers the inputs will be directly pass to the outputs. Therefore, the degradation problem in training the very deep neural networks is greatly alleviated.

In this project, a residual network with 50 layers, namely ResNet50, will be used. Based on the very good performance of this architecture in successfully classifying objects with as many as 1000 different categories on the ImageNet dataset, it should be deep enough to be able to extract the high-level features related with the categories in this project. At the same time, it is more "light-weighted" compared to many other CNN architectures like VGG19 with significantly fewer trainable parameters.

The computational cost can be further reduced by applying the transfer learning technique, which utilizes the pre-trained weights on the "ImageNet" dataset. Although the categories of the "ImageNet" dataset and the dataset from the E-commerce platform Cdiscount are different, the image classification tasks on both datasets can share rather similar low- and middle-level feature filters like edges and certain patterns. In addition, there are considerable overlap of categories between the datasets such as device, fabric, and musical instrument. Therefore, in this project the pre-trained "ImageNet" weights of the ResNet50 network until the last fully-connected layer will be used as the starting point for the subsequent training and fine-tuning.

As mentioned in the Problem Statement section, in a first step the pre-trained weights will be fixed and only the weights of last fully-connected layer will be trained. The pre-trained ResNet50 is used as a feature-transforming tool to extract the high-level bottleneck features. The training of the fully-connected layer can be considered as performing logistic regression classification based on the previously extracted bottleneck features, which can be considered as structured dataset. Similarly, we can also use other well-tested machine learning algorithms for structured data to perform the categorization based on the transformed bottleneck features of training images. XGBoost is such a very promising algorithm, which combines the very good model performance inherited from highly optimized boosted tree methods with its excellent execution speed due to the well-designed system such as parallelization and distributed computation, out-of-core computation and cache optimization.

XGBoost is based on gradient tree boosting developed by Friedmann [6], which is also known as the gradient boosted regression tree. The so-called regression trees are very similar to decision trees, which map an example into its corresponding end leaf node through a series of splitting in the tree structure based on the feature values. However, the output of a regression tree is normally a scalar value associated with the specific leaf node, which is also called the weight of the leaf ($w_{1,2,3}$ in Figure 4).

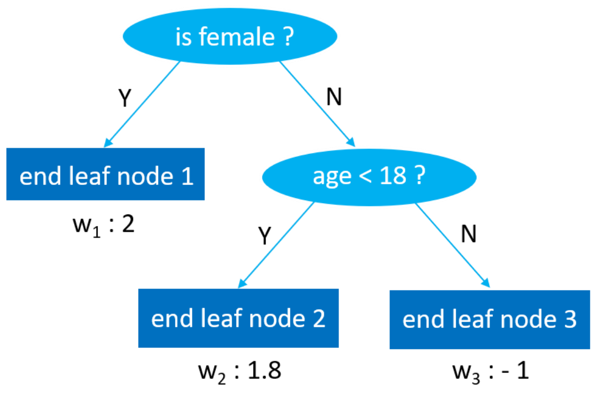


Figure 4. Example of regression tree to predict survival on the Titanic based on the information about passengers.

XGBoost is an ensemble method, which involves training of multiple regression trees. By aggregating the outputs of all its component trees, improved stability and accuracy of the model can be achieved together with reduced variance and overfitting:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)\,, \qquad f_k \in F \tag{3}$$

where $\hat{y}_i$ is the final output of the ensemble classifier with K regression trees for example i with its features x_i, $F = \{f(x) = w_{q(x)}\}$ $(q: \mathbb{R}^m \to T,\ x \epsilon\ \mathbb{R}^m, w\ \epsilon\ \mathbb{R}^T$, m is the number of features, T is the number of leaf nodes of the regression tree). For binary classification, the probability of being a positive instance can be interpreted from $\hat{y}_i$ through logistic regression:

$$P_i = \frac{1}{1 + e^{-\hat{y}_i}} \tag{4}$$

For multiclass classification as in our case, the one-vs-all scheme is applied, which means for each class a corresponding ensemble classifier with K trees will be trained. The probability of being an instance of a certain category c can be calculated based on the outputs of the ensemble classifiers for all the categories through a softmax function:

$$P_{i,c} = \frac{e^{\hat{y}_{i,c}}}{\sum_{n=1}^{N} e^{\hat{y}_{i,n}}} \tag{5}$$

Where $\hat{y}_{i,n}$ is the output of the ensemble classifier for category n.

For each ensemble classifier, the optimum set of the K regression trees are built one after another in K rounds using additive training. In each round, a new tree will be formed, and its output will be added to the output of the previous ensemble:

$$\hat{y}_i(t) = \hat{y}_i^{(t-1)} + \eta \cdot f_t(x_i) \tag{6}$$

where $f_t(x_i)$ corresponds to the regression tree to be built at the t-th round, η is the so-called shrinkage factor, which reduces the influence of each individual tree and leaves space for future trees to improve the model. The regression tree for the t-th round is built in a way so that the following objective will be minimized:

$$\mathcal{L}^{(t)} = \sum_i l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) \tag{7}$$

where l is a differentiable convex loss function that measures the difference between the prediction and the target, $\Omega(f_t)$ is the regularization term that penalizes the complexity of the model and it is defined as follows:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2 \tag{8}$$

In real application, the loss function l will be approximated by using the second order Taylor expansion to facilitate the optimization process in building up each regression tree.

Benchmark

A vanilla CNN will be used as the benchmark model. It contains 3 convolutional layers, each of which is followed by a max pooling layer with a pool size of 3. The first convolutional layer takes the 180 x 180 x 3 matrices of the images as the inputs. It has 512 filters with a kernel size of 3. In the subsequent two convolutional layers, the number of filters doubles 2 times up to 2048 filters in the last convolutional layer whereas the kernel size remains all the same. After 3 such sets of consecutive convolutional layer and max pooling layer, a 180 x 180 x3 input matrix is converted to a matrix with a shape of 7 x 7 x 2048. Finally, the output of the last max pooling layer is flattened to a one-dimensional array, which is then connected to a fully-connected layer with 5270 neurons. Those neurons have a 'softmax' activation function so that they will output the corresponding possibilities for each category. This vanilla CNN will be trained on the same training set and tested on the same evaluation metrics so that the comparison of performance with my own solution is on objective basis.

III. Methodology

Data Preprocessing

In the original data file (train.bson) downloaded from the Kaggle website, each product of the training set presents as a dictionary. The category IDs of the product can be found with the key "category_id", which is a ten-digit decimal integer (e.g. 1000012712). In the data preprocessing step, the category IDs are mapped in to the category indices that range from 0 to 5269 according to the sequence of the name list of the categories provided by Cdiscount. The name list is organized in a hierarchical way, so that the similar categories are also neighbors in the list.

All the images of the product are stored in the binary format under the key "imgs". In the data preprocessing step, those images are firstly read from the train.bson file by using the Image.open method from the PIL library. Then, they are saved as JPEG files under different folders that correspond to different categories by using the Image.save method from the save library. Then, for each category maximum 300 images will be included in the training set as mentioned in the section of Datasets and Inputs. Much smaller amounts of images in each category will be preserved as the validation and test sets. The ratio between the train, validation and test sets is roughly 8 : 1 : 1 in size.

Before training or testing, the RGB images are firstly loaded into memory as 224 x 224 x 3 matrices by using the preprocessing.image.load_img method of Keras library. The pixel values for each color channel in those matrices now range from 0 to 255. In the next step, the pixel values are rescaled between -1 and 1 through the following code:

```
X /= 127.5
X -= 1.
```

where X is the 224 x 224 x 3 Numpy array of pixel values for a certain image. The corresponding category indices of the images are encoded by using the one-hot encoding scheme:

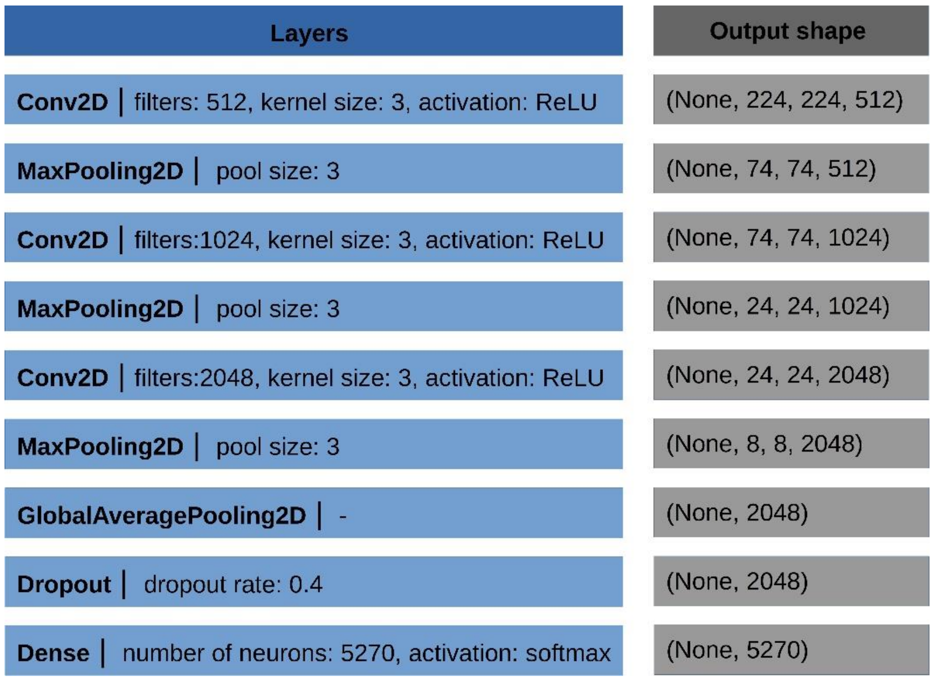
```
lable = [0] * 5270
lable[i] = 1
```

where i is the category index of the product. After this preprocessing step, the matrices of the images and their labels will be fed in to CNN models as the inputs.

Implementation

I. Train the benchmark model

In the configuration and training of the benchmark model, Keras with a TensorFlow backend was used to set up the CNN models. For the benchmark model, firstly a sequential model is created by instantiating the Sequential class by using keras.models.Sequential(). Then three sets of a 2-D convolutional layer and a subsequent 2-D max pooling layer are added in turn to the sequential model, which are then followed by a 2-D global average pooling layer, a dropout layer and finally by a fully connected layer. The architecture and detailed configuration are depicted in Figure 5.



Layers	Output shape
Conv2D | filters: 512, kernel size: 3, activation: ReLU	(None, 224, 224, 512)
MaxPooling2D | pool size: 3	(None, 74, 74, 512)
Conv2D | filters:1024, kernel size: 3, activation: ReLU	(None, 74, 74, 1024)
MaxPooling2D | pool size: 3	(None, 24, 24, 1024)
Conv2D | filters:2048, kernel size: 3, activation: ReLU	(None, 24, 24, 2048)
MaxPooling2D | pool size: 3	(None, 8, 8, 2048)
GlobalAveragePooling2D | -	(None, 2048)
Dropout | dropout rate: 0.4	(None, 2048)
Dense | number of neurons: 5270, activation: softmax	(None, 5270)

Figure 5. CNN architecture used in the benchmark solution.

In the end, the benchmark model is trained for 10 epochs with a batch size of 8 considering the size of the model and the GPU RAM of my computer (8 GB). The weights of the model are saved in hdf5 format after the training

II. Train the stage I model using transfer learning technique

For the proposed solution that applies transfer learning technique, firstly the ResNet50 model with pre-trained "ImageNet" weights are loaded as base model by the following code:

```
base_model = ResNet50(weights='imagenet')
```

Then, the output of its second last layer (namely the layer directly before the final fully connected layer) is redirected to a new fully connected layer with 5270 neurons corresponding the 5270 categories. The model for the transfer learning is established by taking the input of the base model as the input and the output of the newly added dense layer as the output. This is achieved by running the follow code:

```
x = base_model.layers[-2].output

output = Dense(5270, activation='softmax')(x)

model = Model(input=base_model.input, output=output)
```

In a next step, all the layers of the base model are set as untrainable to preserve the low-, middle- and high-level feature filters formed during the previous training on the "ImageNet" dataset. Only the weights of the newly added dense layer will be trained

```
for layer in base_model.layers:

    layer.trainable=False
```

The model is then compiled with the loss function of categorical crossentropy, the Adam optimizer and the accuracy evaluation metrics for the subsequent training:

```
model.compile(loss='categorical_crossentropy',

              optimizer='adam', metrics=['accuracy'])
```

finally, the model is trained for 12 epochs with a batch size of 20.

In a second step, all the weights of the previous ResNet50 model are set as trainable further fine-tuning. In order to do this another model for fine-tuning is created following the same procedure as the previous model but without the step, in which all the layers of the base model are set as untrainable. Then, the previously saved weights are loaded to the fine model:

```
model_fine.load_weights("weights/weights.best.hdf5")
```

Finally, the model is trained for 8 epochs with a batch size of 20 with a much lower learning rate (0.0001).

III. Train the stage II classifiers with the XGBoost

After the training and fine-tuning of this CNN model, the corresponding confusion matrices on both training and validation sets will be calculated. The false positive categories that have occurred at least two times for a certain category in both confusion matrices are systematically listed as shown in

Figure 6. Those lists of so-called easily confusable categories will be stored in a dictionary with the corresponding category indices as the keys for the ease of later access.

Then for each category, a second stage classifier will be trained on a reduced training set, which only consists of the images from the listed false positive categories as well as from the category itself. Python module xgboost is applied to train the second-stage classifiers, which use the bottleneck features of the images extracted from the previous model as the input.

One complication that is worth mentioning is that the bottleneck features of the whole training set should be calculated beforehand to avoid the repetitive calculation in the different sets of easily confusable categories. The calculations of the bottleneck features are done by extracting the output of the second last layers of the previous fine-tuned model:

```
output = model_fine.layers[-2].output
model_bnf = Model(input=model.input, output=output)
```

The calculated bottleneck features are then stored in the corresponding lists of a dictionary, where the category indices are the keys of the lists:

```
bnfs = {}
for x_bnf, y in zip(list_bnfs, list_labels):
    if y not in bnf.keys():
        bnfs[y] = []
    bnfs[y].append(x_bnf)
```

After the training of a complete set of second-stage classifiers for each category, those classifiers are combined with the previous CNN model to provide presumably improved predictions.

The predictions will be made in the two following steps as shown in Figure 7. Firstly, the primitive prediction of the category will be made by the first model. Then, based on the predicted label a corresponding second-stage classifier will be used, which takes the bottleneck feature outputted by the first model as the input. The output of this second-stage classifier is the final predicted category.

Refinement

The hyper-parameters of both the ResNet50 model at stage I and the XGBoost classifiers at stage II were fine-tuned by using the grid search technique. This was achieved by using the sklearn wrappers for Keras and XGBoost (KerasClassifier and XGBClassifier) as well as the GridSearchCV method from sklearn. For the ResNet50 model, the optimal learning rate of Adam optimizer was searched among the following values: [0.00001, 0.000033, 0.0001, 0.00033, 0.001]. For the XGBoost classifiers a grid search of 3 hyper-parameters were performed as specified by the following dictionary: {max_depth:[1,2,3], n_estimators:[5,10,25,50], learning_rate:[0.1,0.3,0.9]}.

The initial setting of learning rate for the ResNet50 model was 0.001. After the refinement step, an optimized value of 0.0001 was used instead, which increased the test accuracy of the model from 0.42 to 0.45. The initial setting of the max_depth, n_estimators and learning_rate of XGBoost classifiers are 1, 25 and 0.1 respectively, which were then optimized as 3, 50 and 0.3 after the hyper-parameter tuning. With the optimized parameters the XGBoost classifier trained on the reduced dataset formed by the easily confusable categories of category 0 showed a decrease of error rate from 0.36 to 0.3.

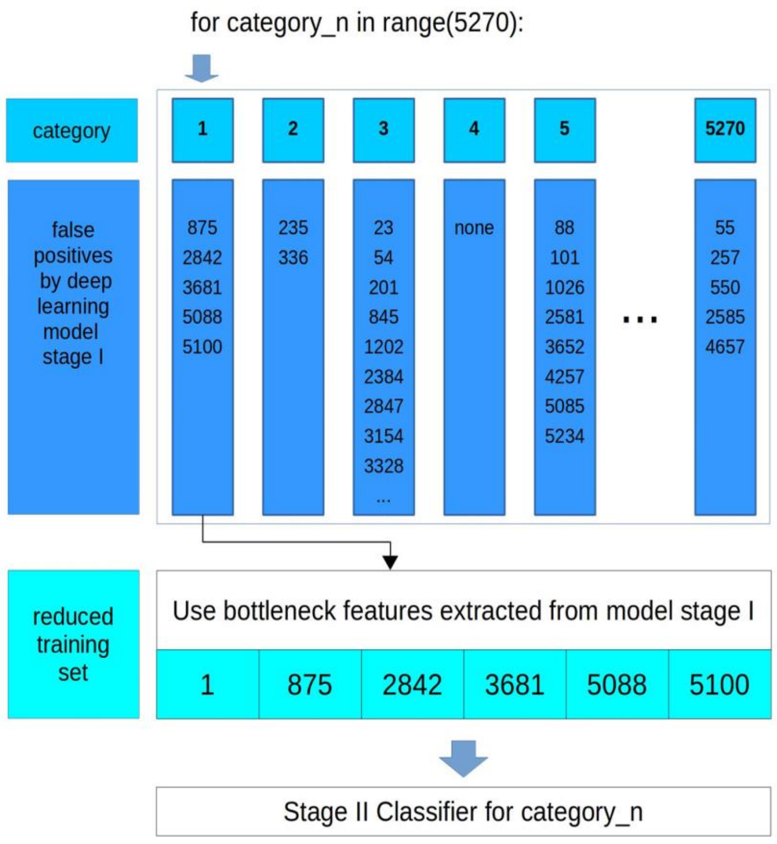


Figure 6. Train the second-stage classifiers for all the categories.

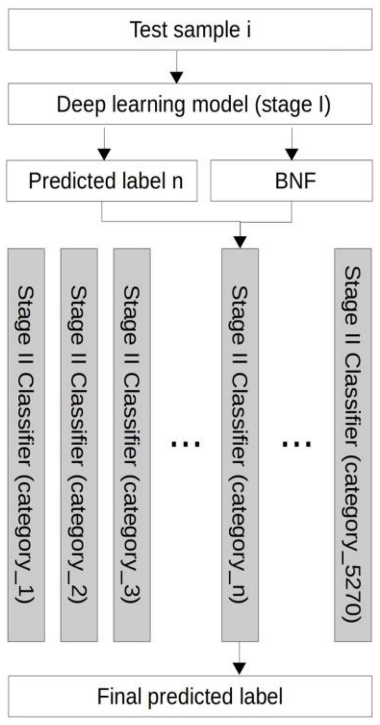

Test sample i
Deep learning model (stage I)
Predicted label n
BNF
Stage II Classifier (category_1)
Stage II Classifier (category_2)
Stage II Classifier (category_3)
...
Stage II Classifier (category_n)
...
Stage II Classifier (category_5270)
Final predicted label

Figure 7. The work-flow of the two-stage model in predicting the category of a product based on images. BNF: bottleneck features.

IV. Results

All the models trained in this project are tested on the same test set, which has been kept away from the training processes from the beginning. The test results are compared in Figure 8 as well as the number of hours spent in training those models. The benchmark model namely the vanilla CNN fails to solve the problem. The test accuracy is only a little better than random guess. Actually it has almost the same number of trainable weights as the ResNet50 model (both of them have around 34M trainable parameters). The reason that the vanilla CNN fails to perform the classification is probably due to its shallow architecture. The vanilla model has only 3 sets of convolutional layers, each of which is followed by a max pooling layers. The small number of layers is insufficient to provide the necessary hierarchy for extracting the high-level features, which, as previously analyzed in the section Definition, are very important to successfully discriminate the very large number of categories and especially the categories that are close in the semantic space.

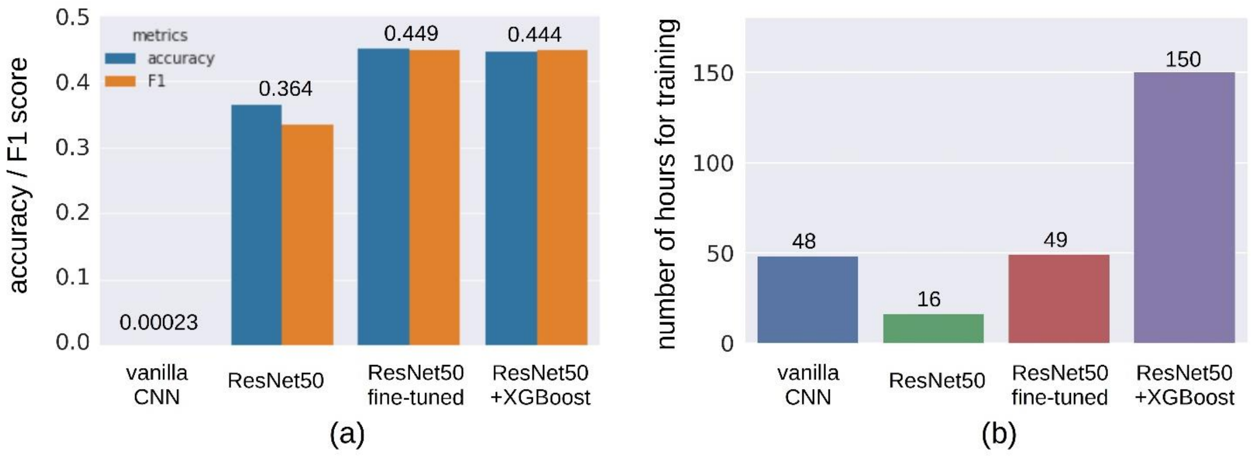


Figure 8. (a) The accuracies and F1 scores of different models on the test set. The numbers displayed above the bars are the respective accuracy values. (b) Number of hours spent in training the models. The vanilla CNN and ResNet50 models were trained on a Nvidia GeForce 1080 GPU and the XGBoost classifiers were trained on an Intel Core i5-7600 CPU.

The proposed solution that applies the transfer learning with the ResNet50 model seems a very effective and efficient approach. Only after two days of training on a Nvidia GeForce 1080 GPU (including the fine-tuning stage), an accuracy and a F1 score of about 45% have been achieved on this large-scale classification problem with 5270 categories. When only the weights of the final fully connected layer of ResNet50 model are trained (without the fine-tuning of the other weights), 80% of the final performance can be achieved within only 1/3 of the original training time (16 hours) compared to the case with fine-tuning. Further training of the fine-tuned model beyond 49 hours has be observed to lead to overfitting and degradation of the model performance.

It was expected that by combining the second-stage classifiers based on XGBoost with the previous stage, namely the fine-tuned ResNet50 model, the performance of the model could be further improved. However, it turned out that with additional 100 hours of training time the accuracy of this 2-stage model is nearly the same as the previous fine-tuned ResNet50 model. In order to find out the effect of adding this stage II classifiers on the category predictions compared to the stage I model (the fine-tuned ResNet50 model), the accuracies for both model in making predictions were recalculated on a category basis. The categorical accuracies of the two models for all the 5270 categories are compared in Figure 9.

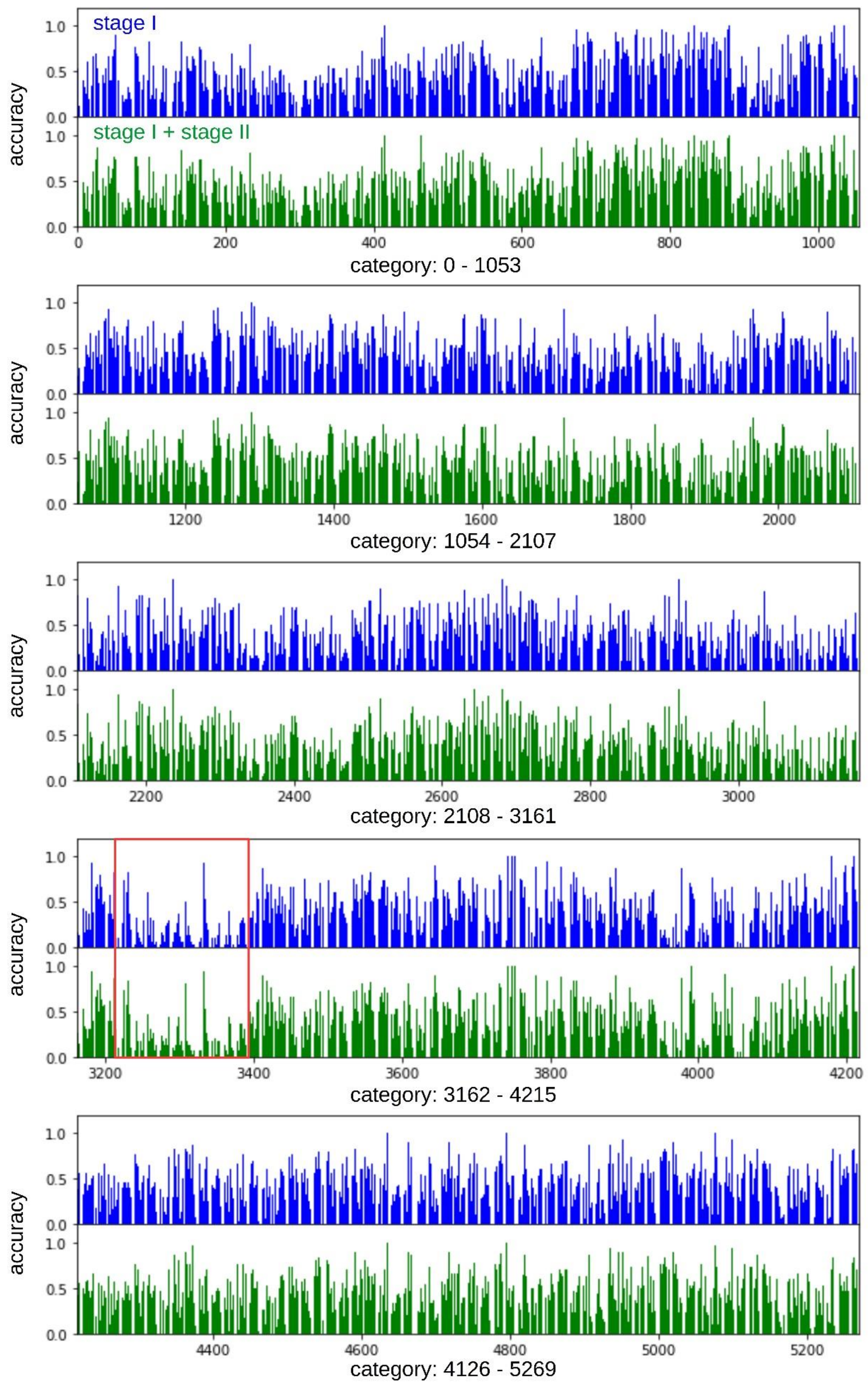


Figure 9. The categorical accuracies of the stage I model (blue) and the complete stage I + stage II model (green) for all the 5270 categories. The x axis are the category indices, which are organized in such a way that the similar categories are also neighbors in the space of indices. The red rectangle marks the different categories of books.

It can been seen from Figure 9 that even on a categorical bases the accuracies of the two models show only negligible differences. It is interesting to note that, the for the 2-stage model, the final decision of the classification was actually made by the XGBoost classifiers, which are non-linear models. In contrast, for the fine-tuned NetRes50 model, the decision was made by the last fully connected layers after the softmax function. The fully-connected layer can be considered as performing a logistic regression based on the bottleneck features, which are linear models. Nevertheless, both the linear and non-linear models based on bottleneck features arrive at nearly the same results. This fact indicates that, during the fine-tuning of ResNet50, the weights under the bottleneck layer have be so optimized that, in the space of the bottleneck features, the different categories are made as linearly separable as possible. Therefore, the classification using a non-linear model like XGBoost based on the bottleneck features probably would not gain additional advantages compared to a simple linear logistic regression.

From Figure 9 it can also be seen that sometimes the categorical accuracies can be continuously low across some ranges of category indices. For example, for those book categories as marked by the red rectangular in Figure 9, the categorical accuracies are significantly lower than other regions. This is actually expected since the image classification performed by CNN, which is based on the graphical features, is unable to extract the lingual information. Only with this additional information such as the title and subtitle of the book as well as other explanatory texts on the covers it is possible to make a good judgment about the contents and thus the types of the book. Besides the book categories, there are also other sets of very similar categories, where an accurate discrimination is extremely hard. Such sets of categories are present as gaps in the spectrum of categorical accuracy that can be frequently seen in Figure 9. The application of the stage II XGBoost classifiers was observed to have on effects in eliminating those gaps, which were, however, designed to further discriminate the easily confusable categories misclassified by the stage I model.

Considering both the performance in prediction accuracy and the time required for training as shown in Figure 7, the fine-tuned ResNet50 model should my final solution model. This model was also tested on 16 real-life images from different E-commerce platforms, which are in various formats, sizes and resolutions, to verify the robustness of the model in real application environment (Figure 10). The predicted categories are listed directly under the images. Their colors indicate the correctness of the prediction (green: correct, yellow: half correct, red: wrong). Two images of this test set (marked by dashed rectangle in Figure 10) actually come from the same product to imitate the common case of the multi-image display of products on E-commerce platforms. One of the images also contains product displayed in an application scenario (the sofa in the third row).

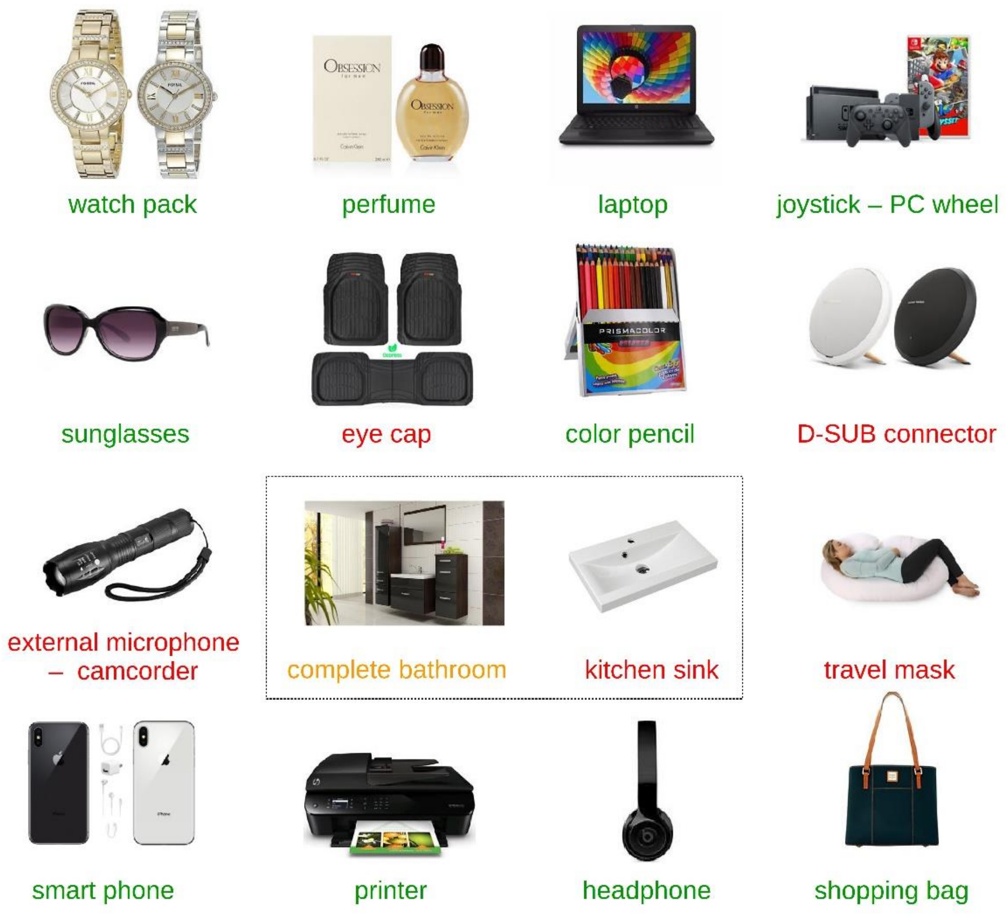


Figure 10. Test the solution model on real-life images from different E-commerce platforms. The predicted categories are shown under the images with their colors indicating the correctness of the prediction (green: correct, yellow: half correct, red: wrong). The dashed rectangle marks the two images that from the same product.

The test shows that the final solution model can make reliable predictions with satisfactory accuracy. Totally 10 out of 16 images are precisely classified. In the misclassified images, some are also hard for human to identify accurately when no additional information is provided, for example the plastic pads for cars (2nd row), which are misclassified as eye cap, as well as the loud speakers with an unusual shape (2nd row), which are misclassified as D-SUB connector. For images that show the application scenarios of products, the model especially tends to be misled. For example, the left image marked by the dashed rectangle shows the application context of a bathroom sink. The semantic classification for the image itself is correct. However, the label is wrong for the product. In the second image of the same product, the bathroom sink was confused by the model with a very similar category: the kitchen sink. However, if the model is able to combine the information from the different images of the same product, it would probably eliminate both types of errors. The last misclassification by the model in the 3rd row is also quite understandable, since the training images of the category travel mask usually contain sleeping people as this image. The sofa in the image also does not have a common shape, which quite resembles a textile pad. Therefore, except for some misleading images showing the application context, the model should be significant enough to solve the problem raised in the beginning.

V. Conclusion

In this project, transfer learning technique together with a well-established deep learning architecture ResNet50 was applied to solve the large-scale classification problem encountered by the E-commence companies. In the data preprocessing step, a proper value for the maximum number of images to be included in the training set was determined. This limit helps to balance the distributions of images among different categories and to prevent from wasting training time on certain extremely abundant categories. Then training images were loaded in to the computer memory as the matrices of pixel values, which were also rescaled to the value range from -1 to 1 before they were fed to the deep learning models. The labels of the images were also mapped into a category index space, where the similarities and neighborhood among categories are considered. In the training phase, in a first step, only the weights of the last fully connected layer of the model were trained and all the other weights remained fixed. In the second step, all the weights of the model are fine-tuned with a much lower learning rate. The solution model shows about 45% accuracy on the test set, which is rather satisfactory considering the large number of categories, the high complexity of the real-life images and the considerable noisiness in both the training and test dataset. The robustness of the model was also verified by using the randomly selected images from different E-commerce platforms.

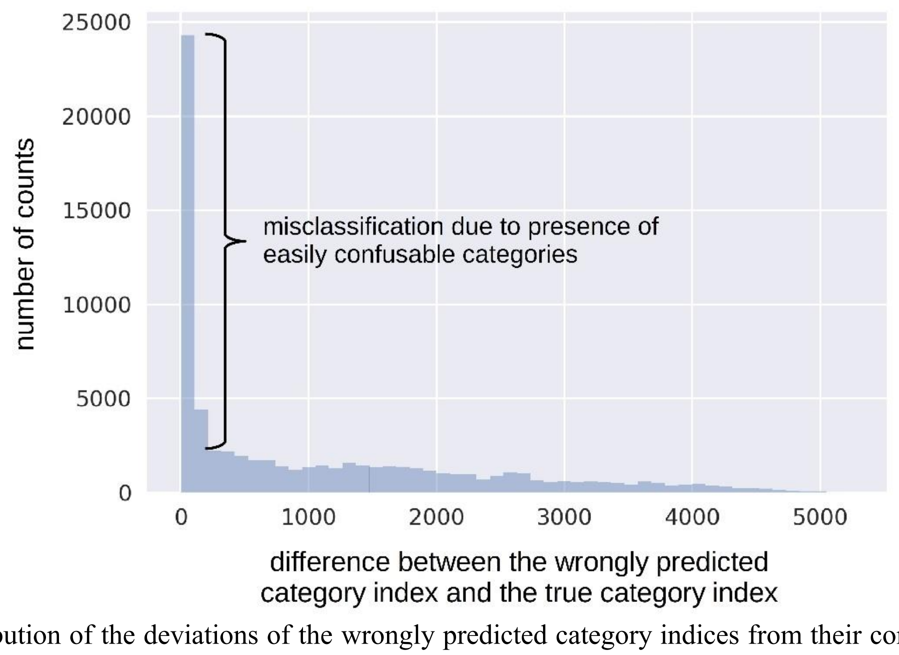


Figure 11. Distribution of the deviations of the wrongly predicted category indices from their corresponding true values on the test set (fine-tuned ResNet50 model). The significantly higher number of counts in the low value range of the divinations shows that considerable misclassification is caused by the presence of similar categories.

The part of the project that I find particularly interesting is the aspect to discriminate the easily confusable categories. Those categories are usually semantically neighboring categories such as the categories of fixed phones and cordless phones as shown in Figure 1. A close examination on the testing results of the solution model shows that a significant amount of misclassification on the test set actually occurs, where the difference between the predicted category index and the true category index is rather small, namely among similar categories (Figure 11). Therefore, further discriminating those easily confusable categories is very meaningful for the enhancing the performance of the model.

Although, the stage II classifiers applied in this project fails to further improve the prediction accuracy, the methodology of learning the sets of the easily confusable categories is still very promising, since 32% of the misclassification on the test set was found to actually fall into the scopes of the easily confusable categories, which were identified for the training of the stage II classifiers. This number is also consistent numerically with the observations shown in Figure 10. If additional information can be utilized in addition to the original bottleneck features of the stage I model such as the text information extracted from book covers, the descriptions provided by the user or even bottleneck features from another model, the previous easily confusable categories could probably be successfully discriminated by the stage II XGBoost classifiers. In addition, the present solution is based on a single image of a certain product. In the future, the information from the multiple images of the product can be synthesized in a proper way and then fed as the input of the stage II classifiers. In this way, instead of misleading the classifier, the images that show the application scenarios can actually provide very useful additional information to further refine the predictions as discussed in the case of bathroom sink in the previous section.

References

[1] Ha, J. W., Pyo, H., & Kim, J. (2016, August). Large-scale item categorization in e-commerce using multiple recurrent neural networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 107-115).

[2] Cevahir, A., & Murakami, K. (2016). Large-scale Multi-class and Hierarchical Product Categorization for an E-commerce Giant. In *COLING* (pp. 525-535).

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

[4] Deng, J., Berg, A. C., Li, K., & Fei-Fei, L. (2010, September). What does classifying more than 10,000 image categories tell us?. In *European conference on computer vision* (pp. 71-84).

[5] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[6] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics* (pp. 1189-1232).