Image Classification
with Convolution Neural Network

1. Data Wrangling

The fashion product data was fetched from Kaggle (Fashion Product Images Dataset | Kaggle). In the original dataset, there are 44k+ product images along with a catalog spreadsheet (styles.csv file) which contains information about products/images, as listed in Table 1 below. The catalog spreadsheet serves as a dictionary for products/images, where we can look up for information such as the product categories, it's display name, the targeted gender, the season the product is intended for, etc.. For my classification part of study, I am only interested in the master categories of products/images. There are 7 master categories, namely, Apparel, Accessories, Footwear, Personal Care, Free Items, Sporting Goods and Home. Apparel, Accessories, Footwear, Personal Care and Free Items are considered as majority categories, with the most products/images (~99.7%) fall under. Free Items, Sporting Goods and Home are considered as minority categories with only 125, 11, 1 products/image falling under, respectively. Since there is only 1 Home product/image, we dropped it from study. Due to time constraint, limited computing power on my local machine, For this study I only studied on a subset of the original Kaggle dataset. The data selection is carried out like the following:

1) from each of the 4 majority categories, I randomly select 1000 product/images.
2) For minority classes, I include all the available product/images. In addition, to bring up minority class representation, I also apply Keras ImageDataGenerator to generate 200 augmented new images which are then combined with the original minority class images to form a bigger dataset. After this step, I end up have 267 images in categories Free Items and 62 in Sporting Good.
3) The datasets obtained from step 1) & 2) are then pooled together to form my study image dataset.
4) For each product/images in the study image dataset, I look up the catalog spreadsheet (styles.csv file) and label it with its matching master category. The labels form the target feature for classification analysis.
5) There is no missing data and no duplicated product/images in the study sample.

2. Data Processing

To understand the data processing steps, I need first briefly describe my modeling strategy. For better model performance and to save computing time and resources, I would like to adopt transfer learning (a popular practice in deep machine learning), and start with a pre-trained convolution neural network (CNN) model, VGG16 in my case. My plan is to sequentially build

up the model like this: the base layer (a pre-trained CNN model VGG16) + two fully connected dense layers + the output layer.

To be fed as inputs to the to-be-built CNN model, each image is processed as the following :

1) resized and converted into a 4D numpy array of shape (4329, 224, 224, 3), using the load_img method from Keras Processing module. The dimension represents 4329 images of size 224×224 in 3 color channels.
2) The array resulting from step 1) is then normalized and transformed by using VGG16 model proprocess_input method that is available in Python module tensorflow.keras.application.

The target feature (a vector of size 4329×1) is processed for training using the to_categorical method from tensorflow.keras.utils module. As the result, it is transformed into an array of shape (4329, 6) representing labels for 4329 images of 6 master categories.

3. Modeling

The train/test data is split at 80/20 ratio. As described in data pre-processing section, my CNN model is a Keras sequential model starting with pre-trainedVGG16 (without top dense layers) as the base layer, and then after flattening it two dense layers are added on top of it. There 50 neurons in the first dense layer and 20 neurons in the second and both layers have relu as the activation function. The last/top/output layer is a dense layer with 6 neurons (for 6 master categories to classify) and softmax as the activation function. The topology and architecture of the CNN model can be viewed in Table 3. I impelment Keras 'adam' optimizer, set 'categorical_crossentropy' as the loss function and choose the classification accuracy from validation data as the metric. When training the model, to accommodate my local machine limited computing capacity I trained with small numbers for epochs and batch size (epochs=10, batch_size=10). The callback of EarlyStopping is used to stop the learning process if there is no accuracy improvement in 5 epochs.

After the model is trained, it is evaluated on the test dataset. The overall accuracy score is 95.85%. A detailed classification report is given in Table 2.

4. Hyperparameter Tuning

Given the good performance of our CNN model, hyperparameter tuning is not really necessary. However, for the learning purpose, I still go ahead and experiment with hyperparameter tuning by applying Keras Tuner module. When training a CNN model, there are quite a few hyaperparameters that can be tuned for model improvement, including (but not limited to) the number of neurons and/or the activation function at a hidden layer, the optimizer ('adam', 'sgd', 'rmsprop'…), the learning rate (e.g., 0.01,0.001…), the training batch size, and the number of epochs. The number of hidden layers of a CNN model can be tuned as well. As we know well that different layers can affect the accuracy. Fewer layers may cause underfitting whereas too many layers may lead to overfitting.

I only experiment with tuning three hyaperparameters: the number of neurons in the first dense layer (the immediate layer added onto the VGG16 base layer), the learning rate of the optimizer and the best number of epochs. The tuning is done in two steps. First, I set the number of epochs fixed (=5) and tuned the first two hyperparameters, with the RandomSearch technique from Keras Tuner module. The search space is set to be small for fast training/tuning purpose, with the number of neurons and the learning rate each allowed to loop over three values, (20, 40, 50) and (0.01, 0.001, 0.0001) repetively. After 6 trails (a pre-determined number), the best configuration is found to be: the number of neurons = 20 and the learning rate=0.001. Next, I re-build my CNN model with the optimal hyperparameters tuned in the earlier step, but this time let the number of epochs=50. By doing this, my goal is to identify the optimal epoch number (epochs=36)

Then finally, my CNN model is rebuilt again with the three tuned hyperparameters. The model is trained and evaluated again. Table 4 gives the classification report on the test dataset. The results from the tuned model actually is worse. Not only the accuracy score decreases from 95.85% (without hyperparameter tuning from Section 4) to 94.47% (with hyperparameter tuning) ,but the minority class 'Sporting Goods" are badly misclassified. In particular, the "Sporting Goods" class is entirely misclassified, as can be seen in Table 4.

5. Addition: Recommendation System built with VGG16

In addition, with the same sample image dataset, I also build a recommendation system. I skipped the data wrangling step as it's already been done in classification. The data pre-processing step is made ready too in classification, except there is no need to worry about the target feature (master categories/labels of images), as we do not need it. Building recommendation system is an unsupervised machine learning.

The recommendation system is built in a similar way as to the classification modeling. I start with a pre-trained VGG16 model (the full model with top dense layers), but for feature extraction this time. Basically by doing this I am summarizing information of each image contained in 224×224×3 pixels into 1,000 newly formed feature. Next on the 1,000 extracted features, I calculate a cosine similarity matrix (of order 4,329×4,329) with entries being pair-wise similarity scores between each pair of the 4,329 images. I then use the similarity scores for recommendations. For example, to find the top 5 recommendations for a given product/image, the 4,328 similarity scores of the product with other products/images are ranked in a descending order. The 5 product/images with the highest 5 similarity scores will be recommended.

6. What's Next

There are quite a few things that interest me and can be further expplored. In the near future, I would like to try to do the following:

1) Move my notebook to Google Colab or Kaggle Notebooks and test the CNN model on the entire 44K+ product/images to see the model performance;

2) Make my model more robust, modular and portable so that it can be readily adopted by other similar image classification tasks.
3) Experiment with tuning other hyperparameters, such as Optimizers ('adam', 'sgd', or 'rmsprop'?), the number of dense layers, and training bacth size…
4) Experiment with adding a dropout layer between two top dense layers to constrain the network from over-learning certain features.
5) Build several different CNN models with other Keras per-trianed models (such as Xception, ResNet50, InceptionV3, etc. in Keras applications) as the base model. Then compare accuracy scores of all models to find the most suitable one for fashion product image classification.
6) Start a new project with a different dataaset from Kaggel to Learn application of deep Neural network in regression.

Appendix

Table 1. Information on Variable in styles.csv File

Variable	Description
ID	The product ID assigned in the product catalog
Gender	Clothing targeted to which gender
MasterCategory	Primary category
SubCategory	Secondary category
articleType	Type of clothing
baseClolor	Descriptive color name
Season	Which fashion season is the product targeted to
Year	Which fashion year is this from
Usage	How is the clothing meant to be used
ProductDisplayName	What is the product title, first word is always the brand name

Table 2. Classification Report

	precision	recall	f1-score
Apparel	0.99	0.98	0.98
Accessories	0.92	0.94	0.93
Footwear	0.99	0.99	0.99
Personal Care	0.99	0.98	0.99
Free Items	0.67	0.70	0.69
Sporting Goods	0.92	1.00	0.96
accuracy			0.96
macro avg	0.92	0.93	0.92
weighted avg	0.96	0.96	0.96

Table 3: Un-tuned Model Topology

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 50)	1254450
dense_1 (Dense)	(None, 20)	1020
dense_2 (Dense)	(None, 7)	147

Total params: 15,970,305

Trainable params: 1,255,617

Non-trainable params: 14,714,688

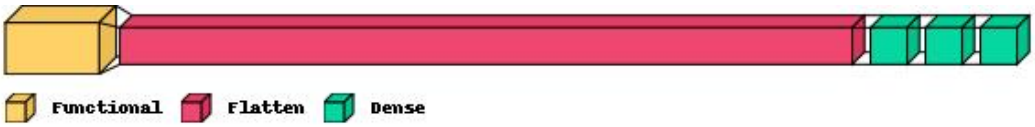

Table 4. Hyperparameter Tuned CNN Model Classification Report

	precision	recall	f1-score
Apparel	0.99	0.98	0.98
Accessories	0.90	0.94	0.92
Footwear	1.00	0.99	0.99
Personal Care	0.99	0.98	0.99
Free Items	0.55	0.66	0.60
Sporting Goods	0.00	0.00	0.00
accuracy			0.94
macro avg	0.74	0.76	0.75
weighted avg	0.94	0.94	0.94

Figure 2. Un-tuned CNN Model Accuracy Plot

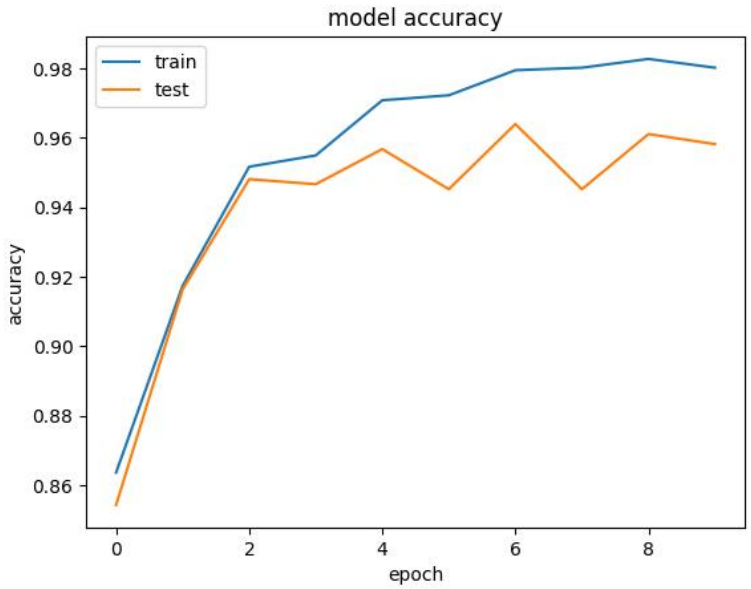

Figure 3. Un-tuned CNN Model Loss by Number of Epoch