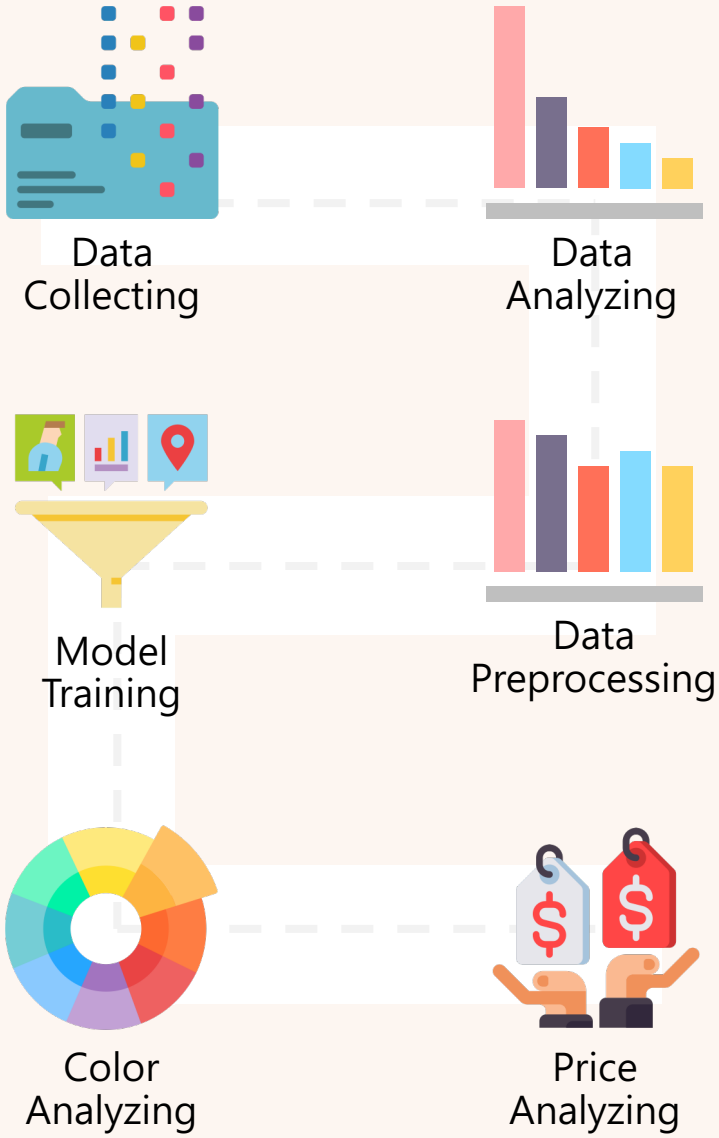
PROJECT

CNN-Based E-commerce Recommender

Teammates :
歐陽宇珊 · 劉柏毅 · 陳秉洋

Process


Data
Collecting
Data
Analyzing
Model
Training
Data
Preprocessing
Color
Analyzing
Price
Analyzing

Flipkart Products

20,000 products on Flipkart

PromptCloud

About this dataset

This is a pre-crawled dataset, taken as subset of a bigger data set (more than 5.8 million products) that was created by extracting data from Flipkart.com, a leading Indian eCommerce store.

Column we used

- image url
- category
- discounted price

image url

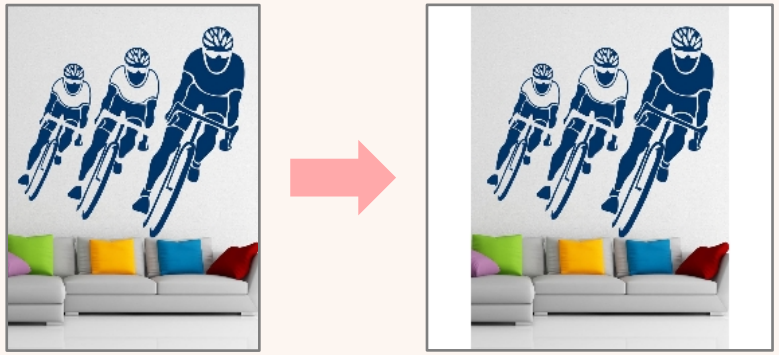
Problem 1 1,554 broken url

Left with 20,000 – 1,554 = 18,446 valid data

Problem 2 Different Shapes and Sizes

Solution

(1)



(2)

256

256

category

Problem 1 Gray Areas among Classification

eg : Home & Kitchen vs Kitchen & Dining

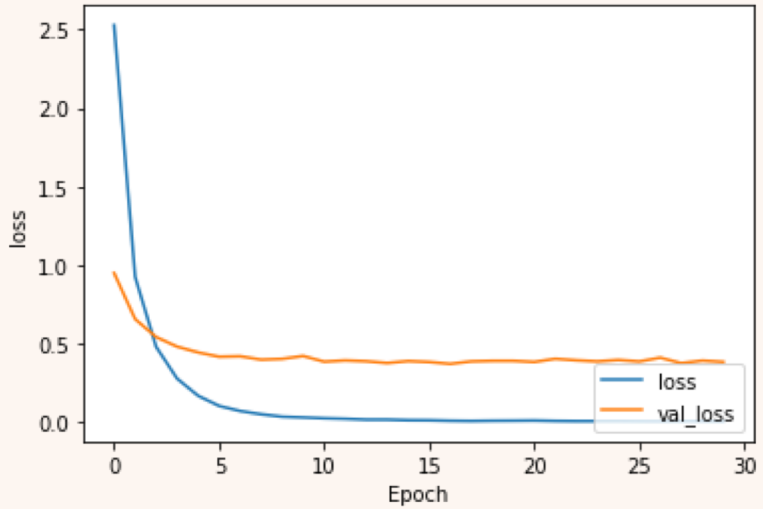
Solution Combine them

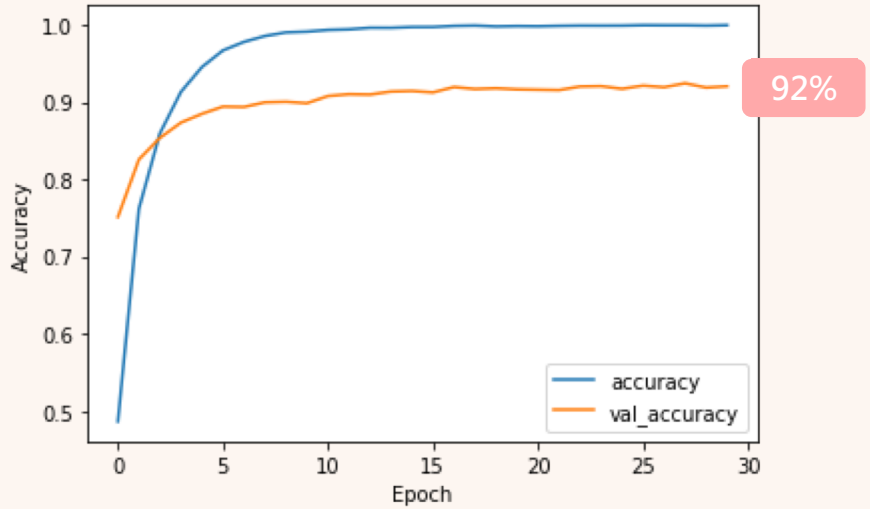
category

Problem 2 Missing or Invalid Value for Classification labels

Counted for 301 of all data

Solution Build a supervised learning model to classify them





category

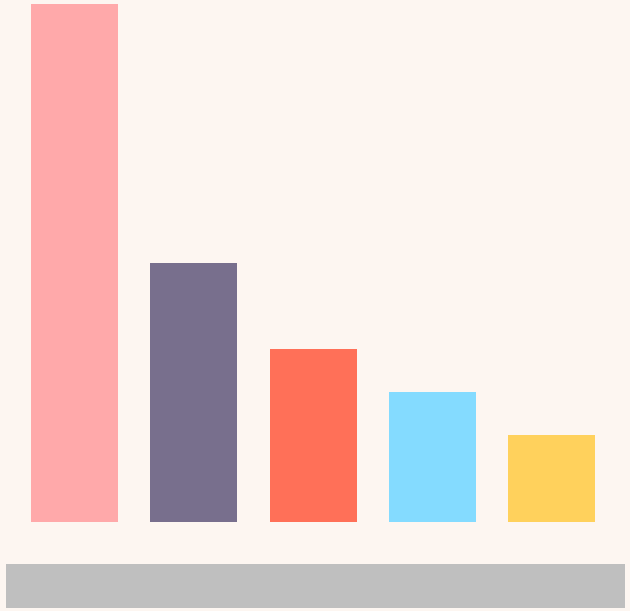
註：資料集的category是階層式的，此處我們僅參考「最上層」。

Solution　　Only 20 types of Categories left after above preprocessing

About the model

Train/Validate	17200 for training / 4300 for validation
Environment	Google Colab GPU, Python, Tensorflow
Criteria	Accuracy
Loss Function	Categorical cross entropy
Optimizer	Adam
Epoch	30
Batch Size	40
Training Time	2.5 hr

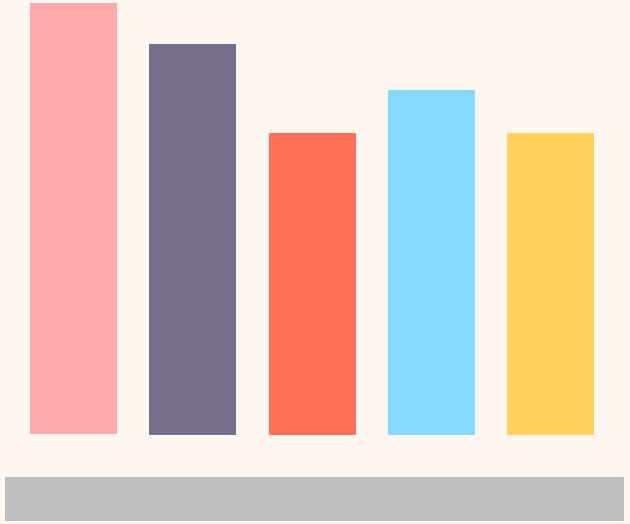
Data Augmentation



Type of Labels 200+

Range of Sample Size 5000+MAX.

Amount of Data 18446



Type of Labels 20

Range of Sample Size 747 MAX.

Amount of Data 21500

Data Augmentation

Scraping

from Amazon

Rotation

Flip

Color Tuned

Random Undersampling

Data Augmentation

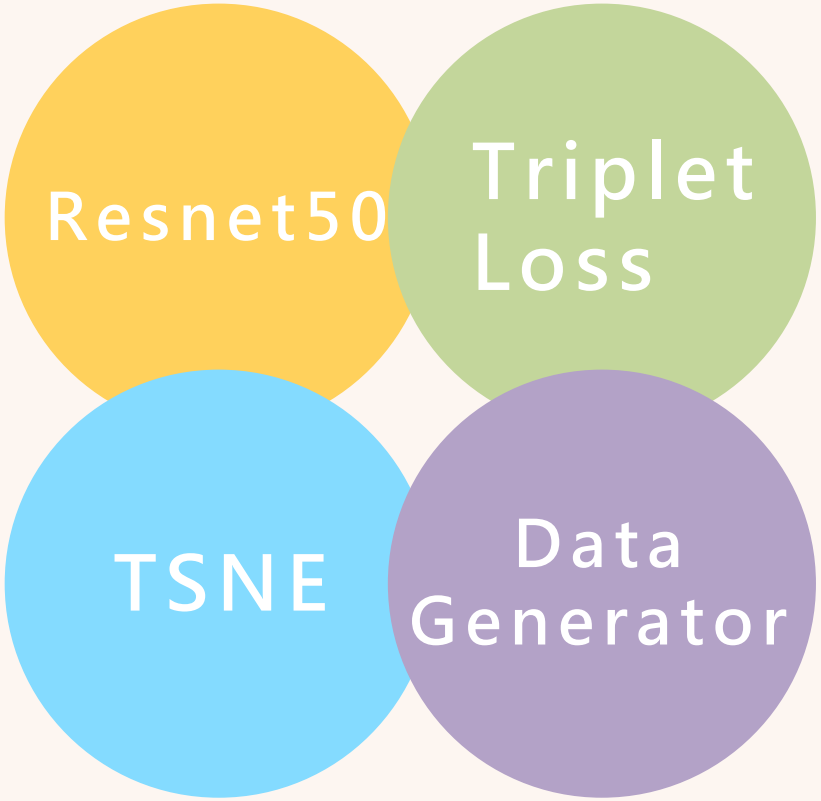
處理前

	labels
Clothing	5167
Jewellery	3546
Footwear	1240
Mobiles & Accessories	1101
Automotive	1012
Home Decor & Festive Needs	946
Home Furnishing	925
Beauty and Personal Care	763
Kitchen & Dining	690
3C	634
Tools & Hardware	472
Baby Care	454
Toys & School Supplies	331
Pens & Stationery	315
Bags, Wallets & Belts	275
Watches	227
Sports & Fitness	186
Cameras & Accessories	82
Eyewear	50
Pet Supplies	30

處理後

	label
Jewellery	1500
Clothing	1500
Kitchen & Dining(aug)	1342
Bags, Wallets & Belts(aug)	1325
3C(aug)	1264
Footwear	1227
Mobiles & Accessories	1099
Sports & Fitness(aug)	1090
Cameras & Accessories(aug)	1088
Watches(aug)	1020
Automotive	1012
Toys & School Supplies(aug)	990
Tools & Hardware(aug)	944
Pens & Stationery(aug)	939
Home Decor & Festive Needs	929
Baby Care(aug)	902
Home Furnishing	880
Eyewear(aug)	856
Pet Supplies(aug)	840
Beauty and Personal Care	753

Main Model




```
from keras.applications.resnet50 import ResNet50
```

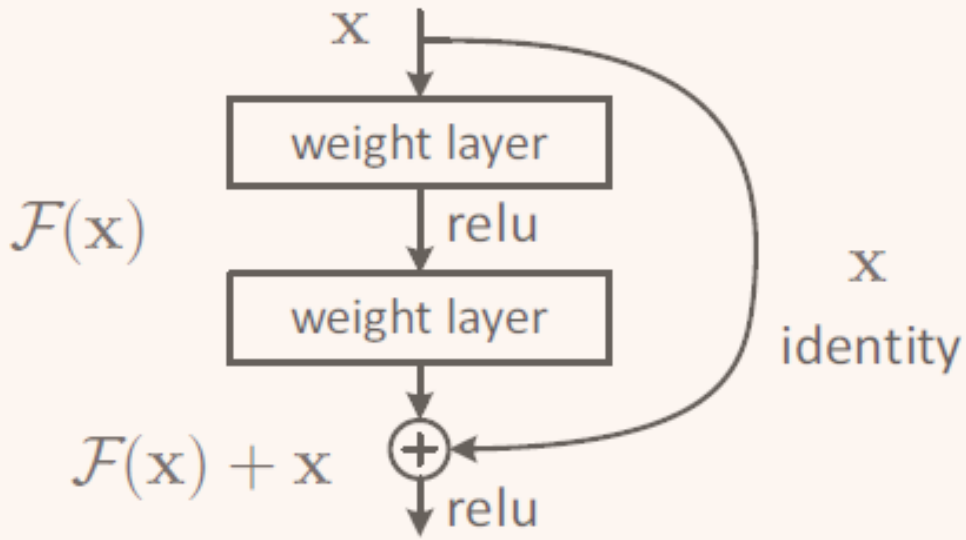


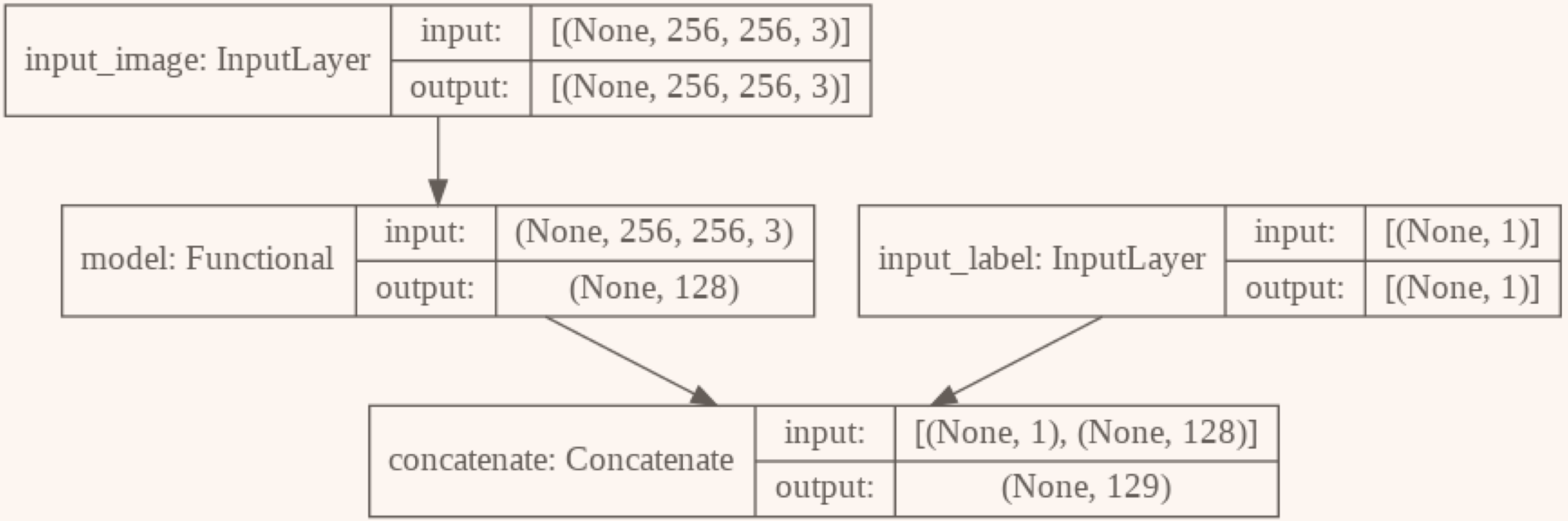
Figure 2. Residual learning: a building block.

Main Model

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Main Model

Embedding



Main Model

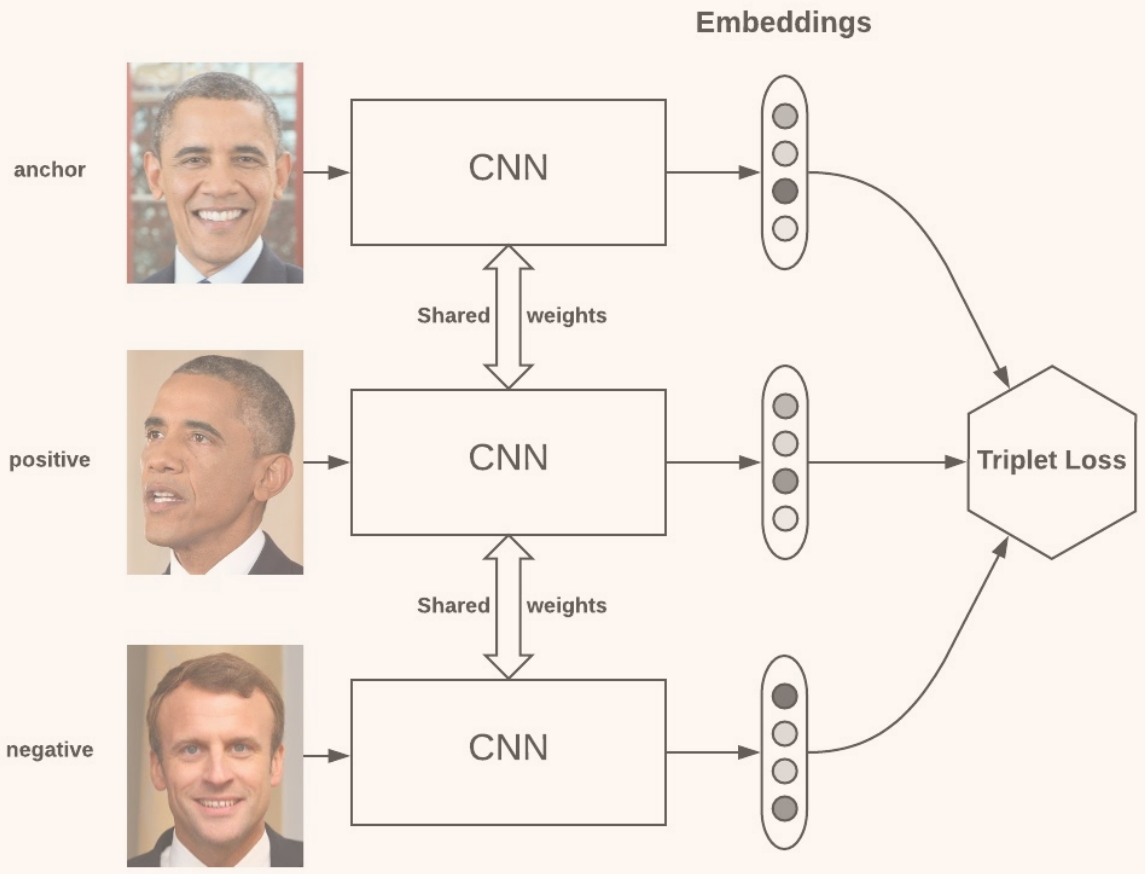
Triplet Loss

- an anchor

- a positive of the same class as the anchor

- a negative of a different class

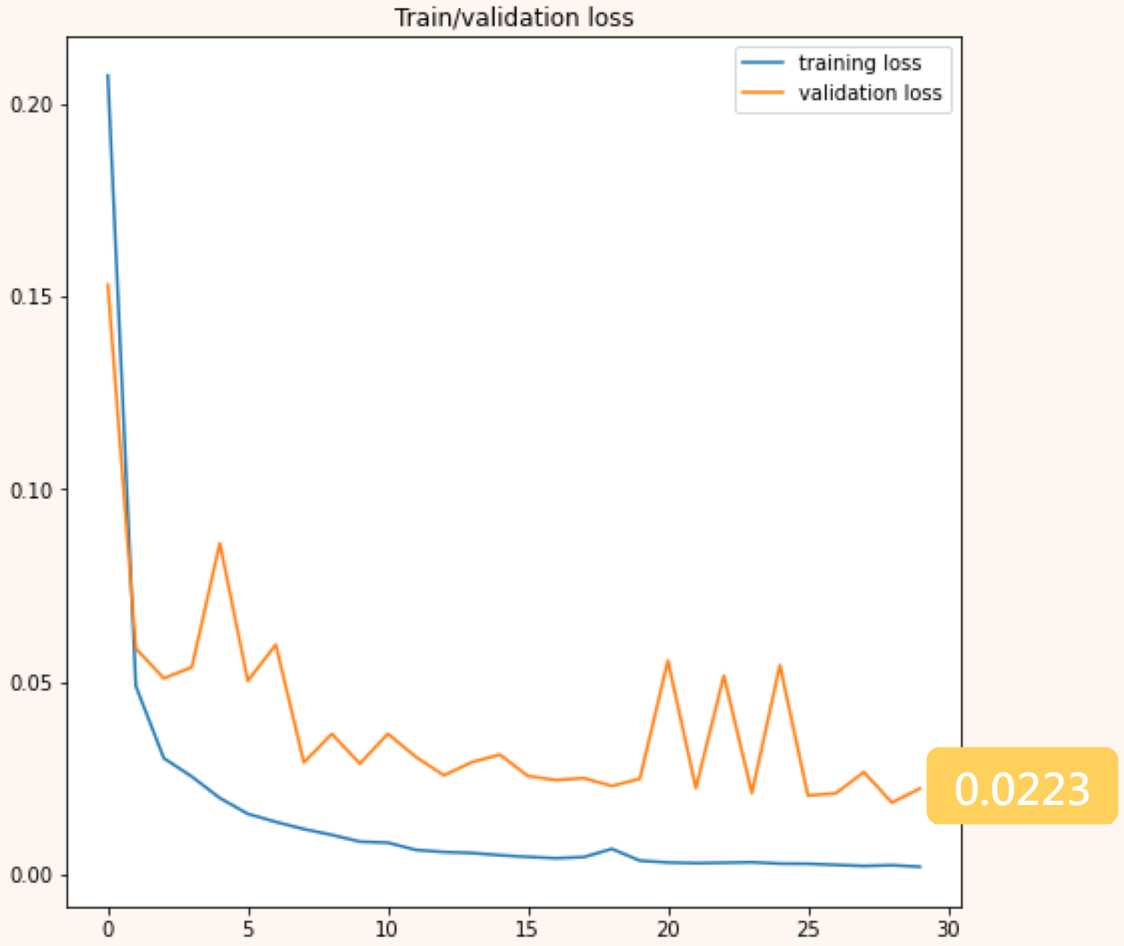
$$\mathcal{L} = max(d(a, p) - d(a, n) + margin, 0)$$



Main Model

About the environment

Train/Validate	17200 for training / 500 for validation
Environment	Google Colab GPU, Python, Tensor Flow
Criteria	Loss
Loss Function	Triplet Loss
Optimizer	Adam (learning rate: 1e-5)
Epoch	30
Batch Size	30
Training Time	5 hr

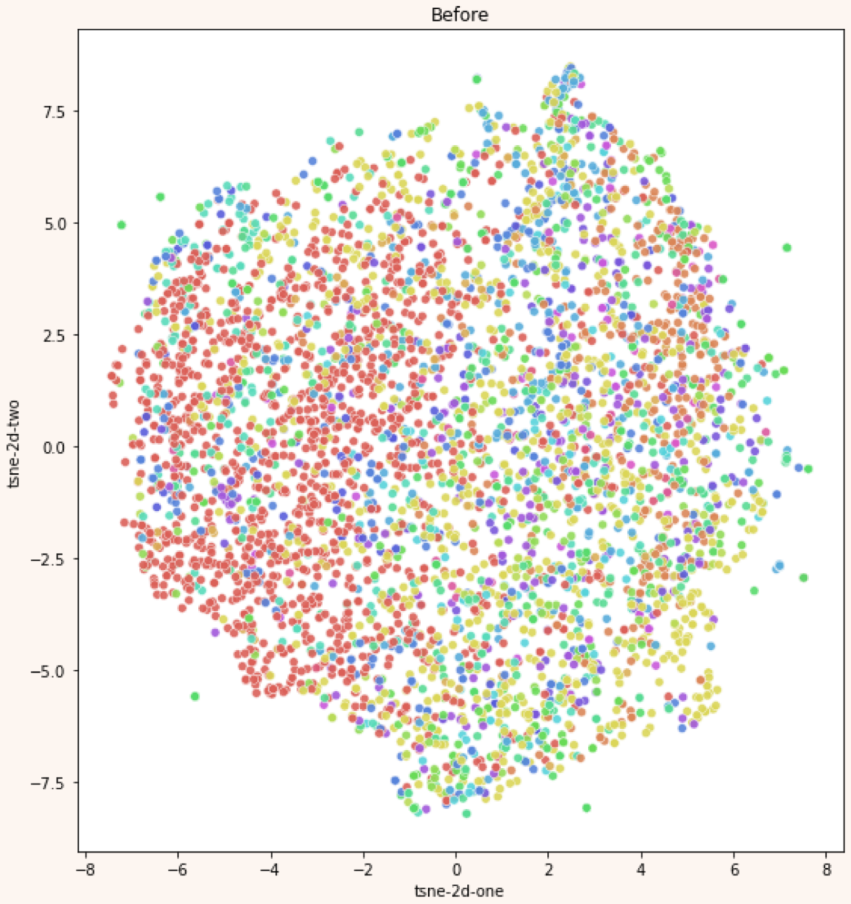


Main Model

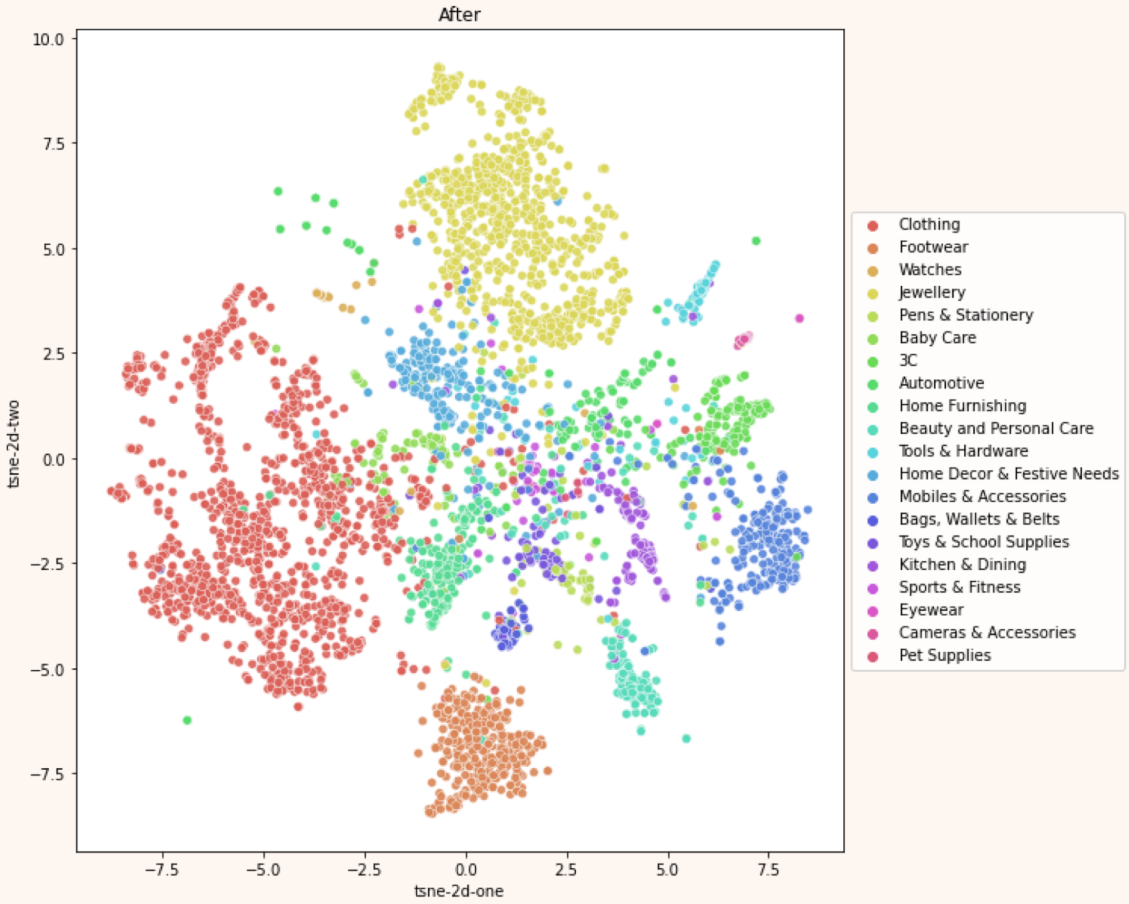

Nonlinear Dimension Reduction
Maintain Partial Structure
TSNE
Randomness
Not suitable for new data

Main Model

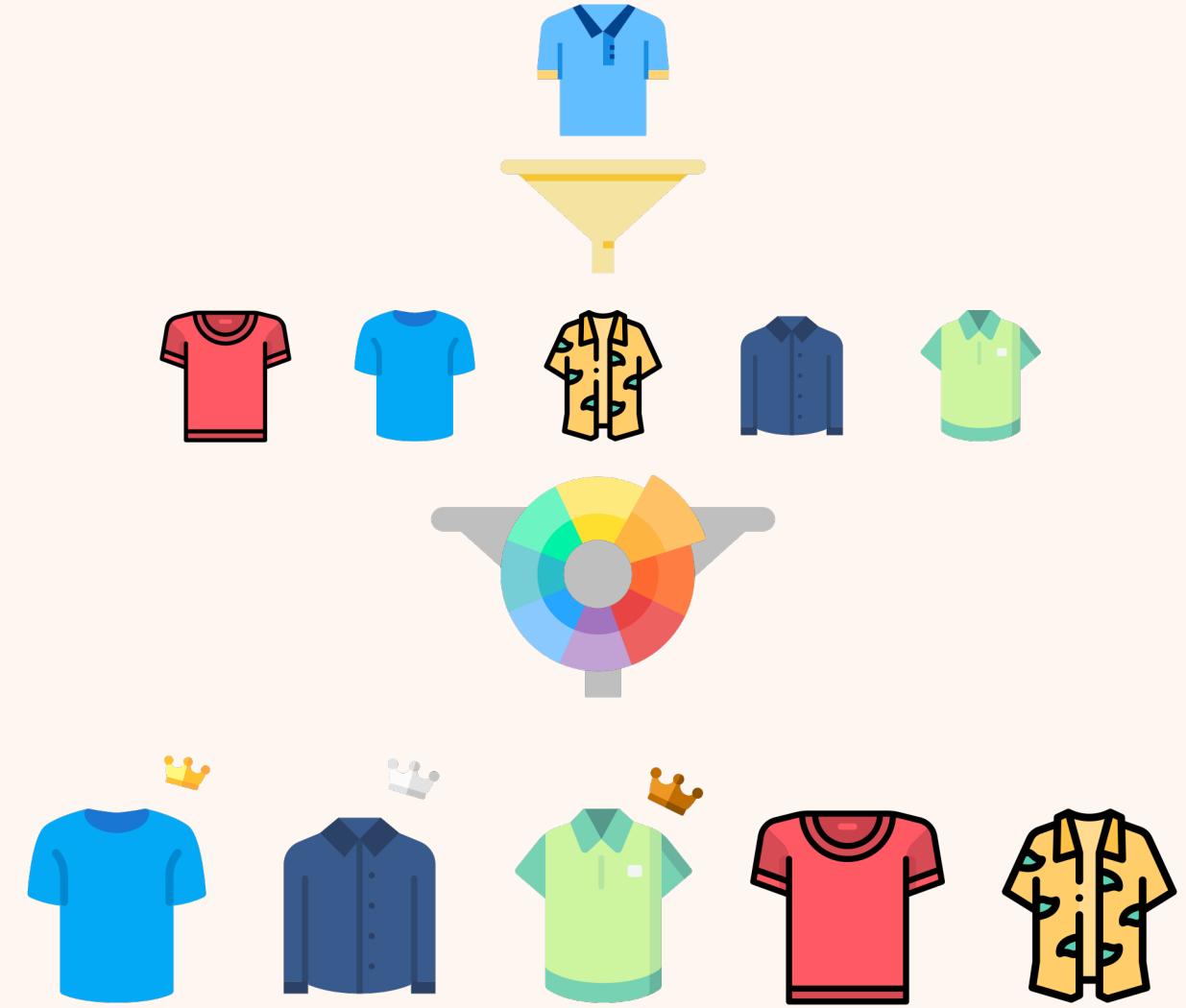
Before Training



After Training



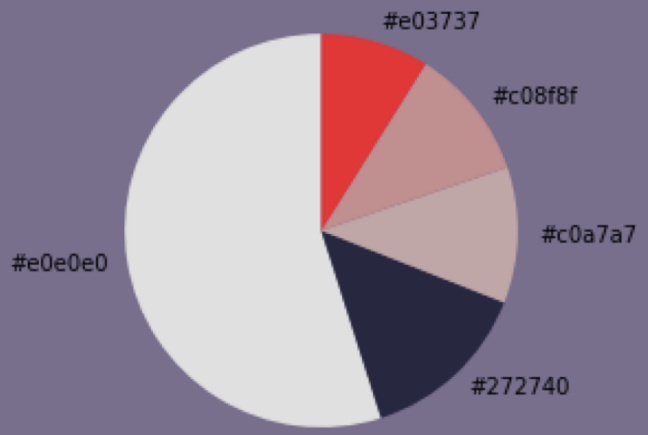
Color Analyzing



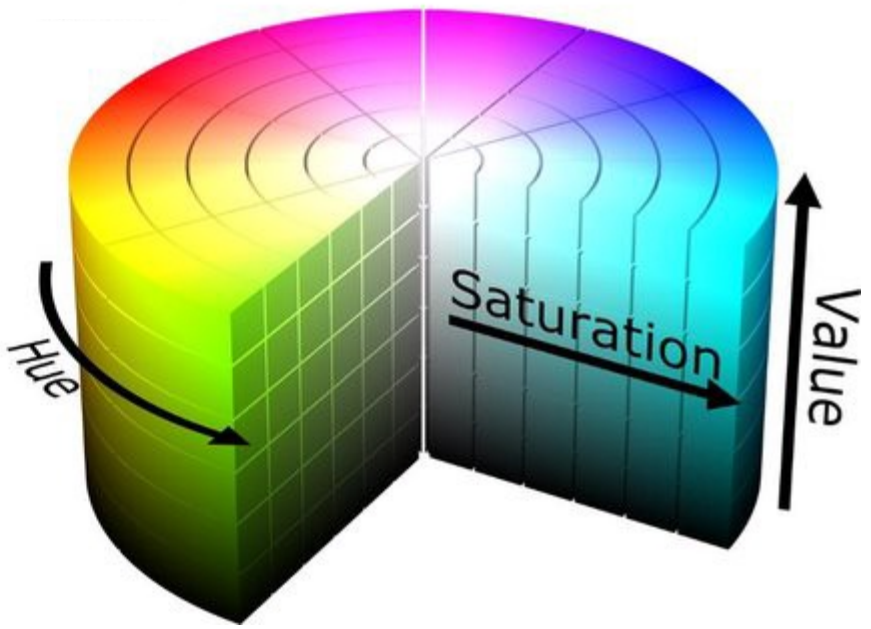
Color Splitting

```
final_img = split_color(imgHSV, (12,8,8))
```

Split the image into 12x8x8 different colors.

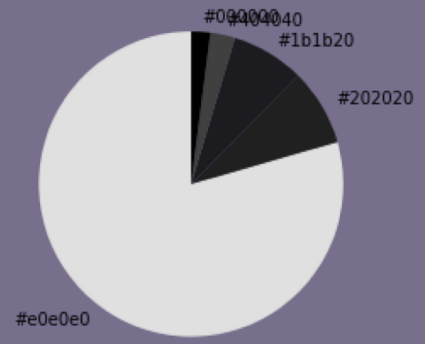


TOP5

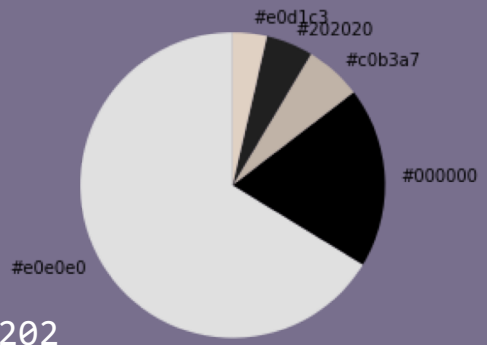


HSV

A

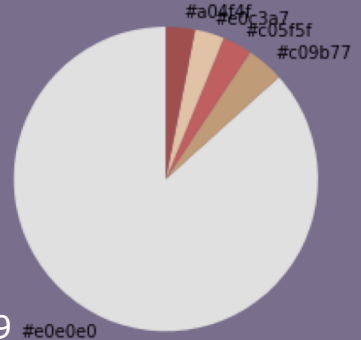


B



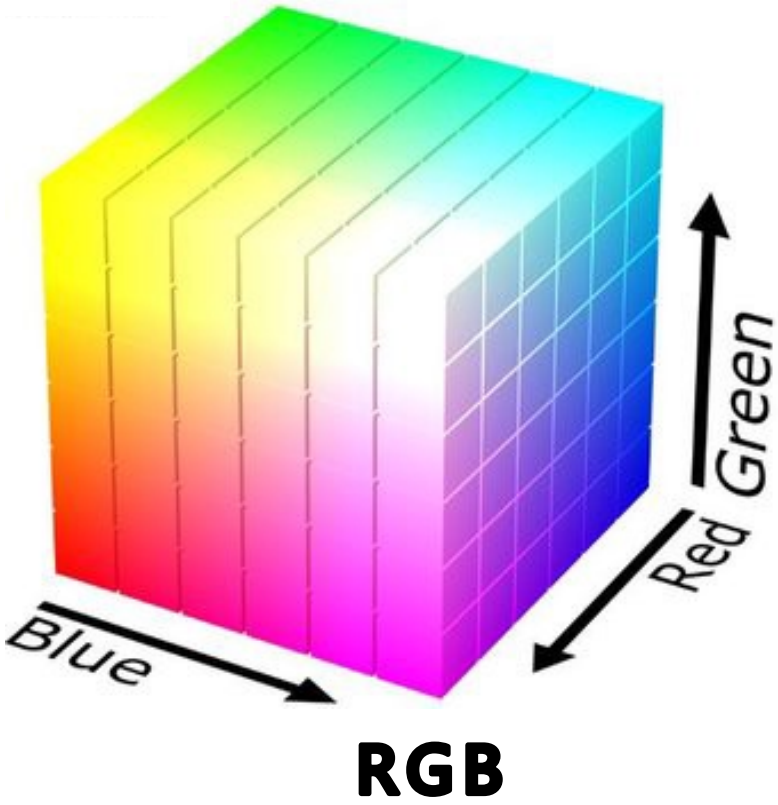
distance: 4.202

C



distance: 18.409

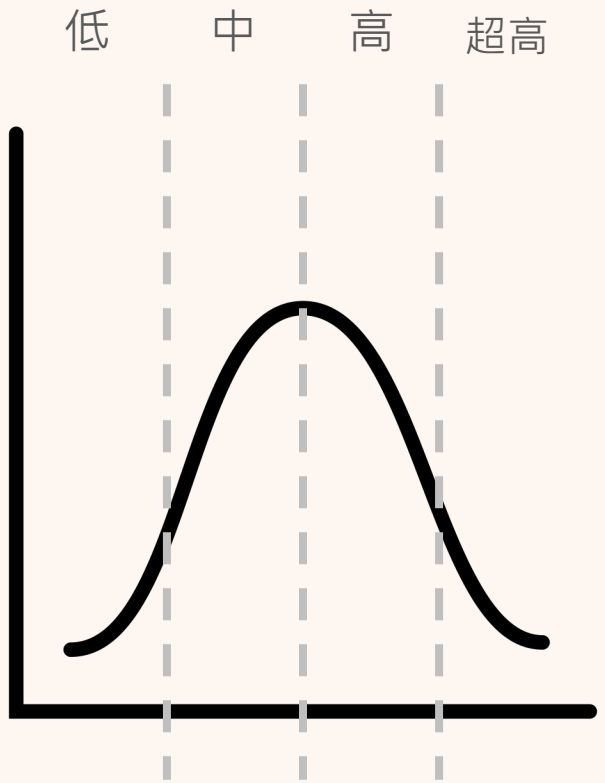
Distance Counting



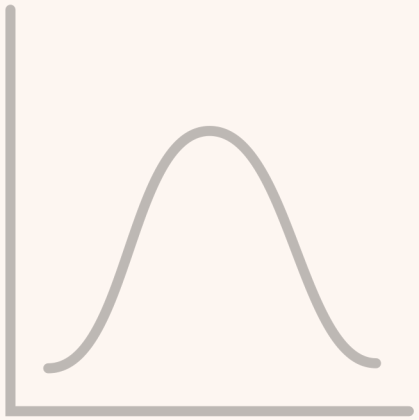
RGB

Price Normalization

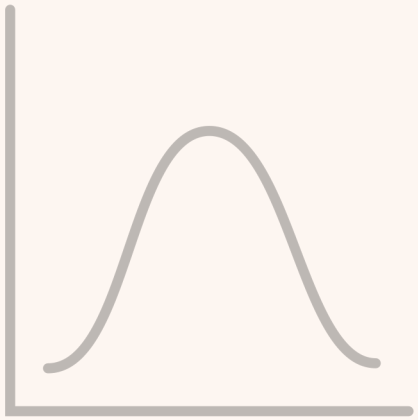
Category A



Category C

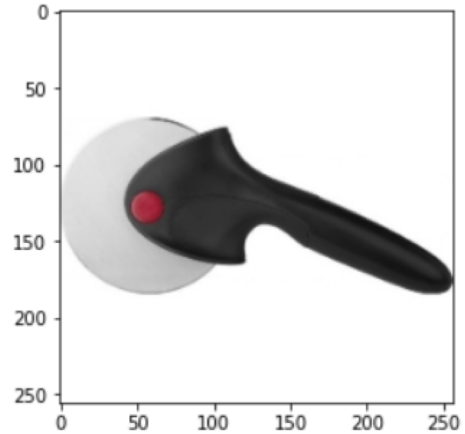


Category B



Thoughts : People who buy cheap clothes may prefer cheaper products

DEMO TIME



Price: 3399.0

7332

Price: 649.0

4069

Price: 959.0

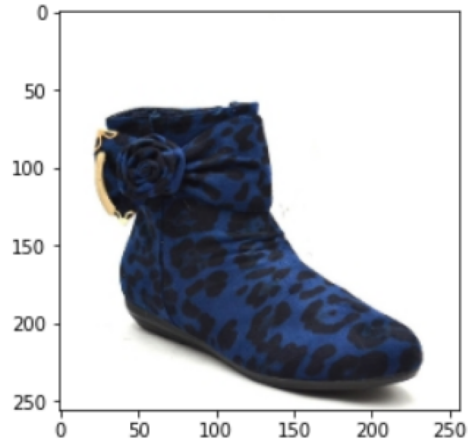
4289

Price: 589.0

4535

Price: 319.0

4602



214

Price: 2239.0

234

Price: 719.0

229

Price: 1099.0

277

Price: 1299.0

316

Price: 299.0

6580

Price: 299.0

5645

Price: 299.0

7019

Price: 349.0

5914

Price: 315.0

5555

Price: 351.0

Annex

Image resizing

```
def img_to_square(img, side_length=256):
    h, w, d = img.shape
    WHITE = [255,255,255]
    if h > w:
        new_h = side_length
        resized = cv2.resize(img,(int(w*new_h/h),new_h),interpolation=cv2.INTER_CUBIC)
        left_border = right_border = (new_h-int(w*new_h/h))//2
        if (new_h-int(w*new_h/h))%2 != 0:
            right_border += 1
        return cv2.copyMakeBorder(resized,0,0,left_border,right_border,cv2.BORDER_CONSTANT,value=WHITE)
    elif h < w:
        new_w = side_length
        resized = cv2.resize(img,(new_w,int(h*new_w/w)),interpolation=cv2.INTER_CUBIC)
        top_border = bottom_border = (new_w-int(h*new_w/w))//2
        if (new_w-int(h*new_w/w))%2 != 0:
            bottom_border += 1
        return cv2.copyMakeBorder(resized,top_border,bottom_border,0,0,cv2.BORDER_CONSTANT,value=WHITE)
    else:
        return cv2.resize(img,(side_length,side_length),interpolation=cv2.INTER_CUBIC)
```

Image Scraping & Under-Sampling

```
input = 'Home & Kitchen'
keyword = input+'+amazon'
response = requests.get(f’html)
soup = BeautifulSoup(response.text, "html.parser")
results = soup.find_all("img")
image_links = [result.get("src") for result in results]
count=1
for index, link in enumerate(image_links):
   os.chdir(path+'Extra Images')
   if not os.path.exists(input):
       os.mkdir(input)
   try:
     img = requests.get(link)
     with open(path+'Extra Images/'+input+'/'+str(count) + ".jpg", "wb") as file:
       file.write(img.content)
     count += 1
   except:
     pass
```

Image Scraping & Under-Sampling

```
import random
all_labels = list(tables.labels.unique())
#perform undersampling and sort images
for i in all_labels:
 random.seed(12345)
 same_cate_ind = list(tables[tables['labels'] == i].index)
 if len(same_cate_ind) > 2000:
   undersamp_ind = random.sample(same_cate_ind,1500)
   new_sample = images[undersamp_ind]
   np.save(path+'Resampled Np/'+str(i)+'.npy',new_sample)
   print(i,new_sample.shape[0])
 else:
   new_sample = images[same_cate_ind]
   np.save(path+'Resampled Np/'+str(i)+'.npy',new_sample)
   print(i,new_sample.shape[0])
```

Model 1

```
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.optimizers import Adam
# Size
IMAGE_SIZE = (256, 256)
# Classes
NUM_CLASSES = 20
# Pretrained Resnet50
net = ResNet50(include_top=False, weights='imagenet', input_tensor=None,
               input_shape=(IMAGE_SIZE[0],IMAGE_SIZE[1],3))
x = net.output
x = Flatten()(x)
# DropOut layer
x = Dropout(0.5)(x)
# Dense layer
output_layer = Dense(NUM_CLASSES, activation='softmax', name='softmax')(x)
```

Model: Preprocessing

```
# freeze layers
net_final = Model(inputs=net.input, outputs=output_layer)
for layer in net_final.layers[:FREEZE_LAYERS]:
   layer.trainable = False
for layer in net_final.layers[FREEZE_LAYERS:]:
   layer.trainable = True

# tuned hyperparameters
net_final.compile(optimizer=Adam(lr=1e-5),
                  loss='categorical_crossentropy', metrics=['accuracy'])

# output summary
print(net_final.summary())
```

Model: Preprocessing

```
# model training
import time
start = time.time()
epochs = 30
batch_size = 40
history = net_final.fit(X_train, y_train, epochs=epochs,
                  batch_size=batch_size,
                  validation_data=(X_val, y_val),
                  verbose=1)
end = time.time()
print('Time: ',end-start)
```

Data Generator

```
def generator(X_data, y_data, batch_size):
  samples_per_epoch = X_data.shape[0]
  number_of_batches = samples_per_epoch/batch_size
  counter=0
  while 1:
    X_batch = np.array(X_data[batch_size*counter:batch_size*(counter+1)]).astype('float32')
    X_batch = X_batch/255
    y_batch = np.array(y_data[batch_size*counter:batch_size*(counter+1)]).astype('float32')
    dummy_gt_train = np.zeros((len(X_batch), embedding_size + 1))
    counter += 1
    yield [X_batch,y_batch], dummy_gt_train
    #restart counter to yeild data in the next epoch as well
    if counter >= number_of_batches:
        counter = 0
```

Model: Embedding

```
def create_base_network(image_input_shape, embedding_size):
    input_tensor = Input(shape=image_input_shape, name="input")

    model = ResNet50(weights='imagenet', include_top= False ,
                input_tensor=input_tensor,
                input_shape=None,)

    x = model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(embedding_size)(x)

    base_network = Model(inputs=input_tensor, outputs=x)
    # plot_model(base_network, to_file=path+'base_network.png', show_shapes=True, show_layer_names=True)
    return base_network
```

Model: Embedding

```
base_network = create_base_network(input_image_shape, embedding_size)

input_images = Input(shape=input_image_shape, name='input_image') # input layer for images
input_labels = Input(shape=(1,), name='input_label')    # input layer for labels
embeddings = base_network([input_images])               # output of network -> embeddings
labels_plus_embeddings = concatenate([input_labels, embeddings])  # concatenating the labels +
embeddings

# Defining a model with inputs (images, labels) and outputs (labels_plus_embeddings)
model = Model(inputs=[input_images, input_labels],
              outputs=labels_plus_embeddings)

model.summary()
```

TSNE

```
time_start = time.time()
tsne_after = TSNE(n_components=2, verbose=0, perplexity=40, n_iter=300)
tsne_results_after = tsne_after .fit_transform(x_embeddings)
print('t-SNE done! Time elapsed: {} seconds'.format(time.time()-time_start))
```

Price Normalization

```
def per30(x):
    return x.quantile(0.30)
def per65(x):
    return x.quantile(0.65)
def per90(x):
    return x.quantile(0.90)
price_level =
table.groupby('labels')['discounted_price']
.aggregate([per30,per65,per90]).sort_values
('per65',ascending=False)
```

```
def price_tag(x):
    low, middle, high =
price_level.loc[x['labels']]
    tag = 0
    if low == middle == high:
        tag = 2.5
    elif x.discounted_price <= low:
        tag = 1
    elif low < x.discounted_price <= middle:
        tag = 2
    elif middle < x.discounted_price <= high:
        tag = 3
    else:
        tag = 4
    return tag
table['price_tag'] =
table.apply(price_tag,axis=1)
table.price_tag.value_counts()
```

Final Recommender

```
def get_final_recommendation(idx_ref, filter=20, color_weight=1, price_weight=1, top=5):
#get broader recommendation based on CNN
idx_rec, idx_sim = get_recommender(idx_ref, top_n = filter)
#get price and color analysis
price_order, price_df = get_price_order(idx_ref,idx_rec)
total_df = get_color_info(idx_ref, price_order, price_df)
#design scores
# p_score = np.linspace(0,price_weight,filter)
# c_score = np.linspace(0,color_weight,filter)
total_df['price_score'] = (3-total_df['Diff'])/3
# color_df['color_score'] = c_score
# total_df['price_score'] = p_score
#combine two analysis
# total_df = pd.merge(price_df,color_df,on='Index')
total_df['total'] = total_df['price_score']*price_weight +
total_df['color_score']*color_weight
#reset order
total_df = total_df.sort_values(by=['total'],ascending=False)
return total_df.Index[0:top], total_df
```

Visualize the result

```
def plot_figures(figures, nrows = 1, ncols=1,figsize=(8, 8)):
    """Plot a dictionary of figures.

    Parameters
    ----------
    figures : <title, figure> dictionary
    ncols : number of columns of subplots wanted in the display
    nrows : number of rows of subplots wanted in the figure
    """

    fig, axeslist = plt.subplots(ncols=ncols, nrows=nrows,figsize=figsize)
    for ind,title in enumerate(figures):
        axeslist.ravel()[ind].imshow(cv2.cvtColor(np.array(figures[title],dtype="uint8"),cv2.COLOR_BGR2R
GB))
        axeslist.ravel()[ind].set_title(title)
        axeslist.ravel()[ind].set_axis_off()
        axeslist.ravel()[ind].text(0,0,'Price: '+str(table.iloc[int(title),:]['discounted_price']))
    plt.tight_layout() # optional
```

Demonstration

```
# Idx Item to Recommender
idx_ref = Input any number from 0 to 18445

# Recommendations
idx_rec, comparison = get_final_recommendation(idx_ref, filter=20, color=0, price=1, top=9)

# Plot
#====================
plt.imshow(cv2.cvtColor(np.array(x[idx_ref],dtype="uint8"),cv2.COLOR_BGR2RGB))
print('Price: ',table.iloc[idx_ref,:]['discounted_price'])
# plt.imshow(np.array(x_test[idx_ref]*255,dtype="uint8"))
# generation of a dictionary of (title, images)
# figures = {str(i): cv2.cvtColor(np.array(x[i],dtype="uint8"),cv2.COLOR_BGR2RGB) for i in idx_rec}
figures = {str(i): x[i] for i in idx_rec}
# figures = {str(i): np.array(x_test[i]*255,dtype="uint8") for i in idx_rec}
# plot of the images in a figure, with 2 rows and 3 columns
plot_figures(figures, 3, 3)
```