FloorDog

Intelligent Fashion Recommender

Github repostitory: https://github.com/Ethan-syc/FloorDog
FashionNet repository: https://github.com/evanWang1409/FashionNet-Pytorch

Problem Description & Motivation

Fashion is an outlet for us to showcase our identity. We spend considerable time determining what to wear, and how to express ourselves through our clothes. Our appearance serves as our first impression; this makes it critical for us to choose the correct outfit.

Many people struggle to determine which clothes to wear. The fashion industry is constantly changing, making it difficult to keep track of current trends. Our motivation stems from this problem: we decided to create a web application that helps users built their outfits.

Our application has two functions:

1. Provide potential interested clothing items for user by filtering gender, category and attributes. For example, if user select "men", "shirts", "long-sleeve", we will present all clothing items satisfying the requirements.
2. Provides a recommendation of potential compatible items of clothing to help users complete their outfit.

This application is important, because it saves users the trouble of figuring out what to wear. The user just takes a picture of an item of clothing, and our application will give suggestions for compatible clothes based on the color and category.

Web Scraping

We used the HTML parser Beautiful Soup to scrape from www.ssense.com/en-us. SSENSE is an international fashion platform that sells products from designers. We start from the main product page, and access each individual product through its link and store the link as item_url. Each new product is assigned with a unique id among men or women. On each individual product page, we scraped the product's name (clothes_name), description (clothes_detail) and materials (material) . Below the description of each product, there's a part named 'Styled With' that contains links to other products that are compatible with the current product. We store these links as accessory1, accessory2 etc.

We then go over the entire table and scrap the information of the accessories for every recorded clothing item. We repeat this process until all proposed compatible accessories are recorded in our database as new entries. Since the url for every recorded clothing items is unique, we can assign a unique index for every recorded clothing items. After this, for the accessory columns, we no longer need to record them as links, but we can instead store the index of those items in our database.

To get clothes' attributes, like "long sleeve, graphic", we seperate those attributes from the descriptions and materials scrapped for all the clothes in our database. We use a reduced map algorithms to obtain the frequency of every single words and every 2 words (some attributes like "long sleeve" have two words). After that, we find the entries with high frequencies, and then manually filter out the meaningless ones, such as "a, the" et al. The words left with high frequencies are the attributes we recorded in our database.

Database design and website development

In order to power the recommendation neural network as well as filter function that sorts through the database, Django is used to build an interactive and well-designed website. From hundreds of plausible functions that FloorDog is able to power, we implement two essential functions for a fashion recommending website, namely item filtering and upload recommendation.

Database design and implementation

After web-scraping is finished, we went through all our data and designed our database schema, which is written both in create.sql as well as models.py. In essence, we designed four basic tables for the storage and later processing of these data. In total, more than 8700 clothing items of each gender are being stored in our database.

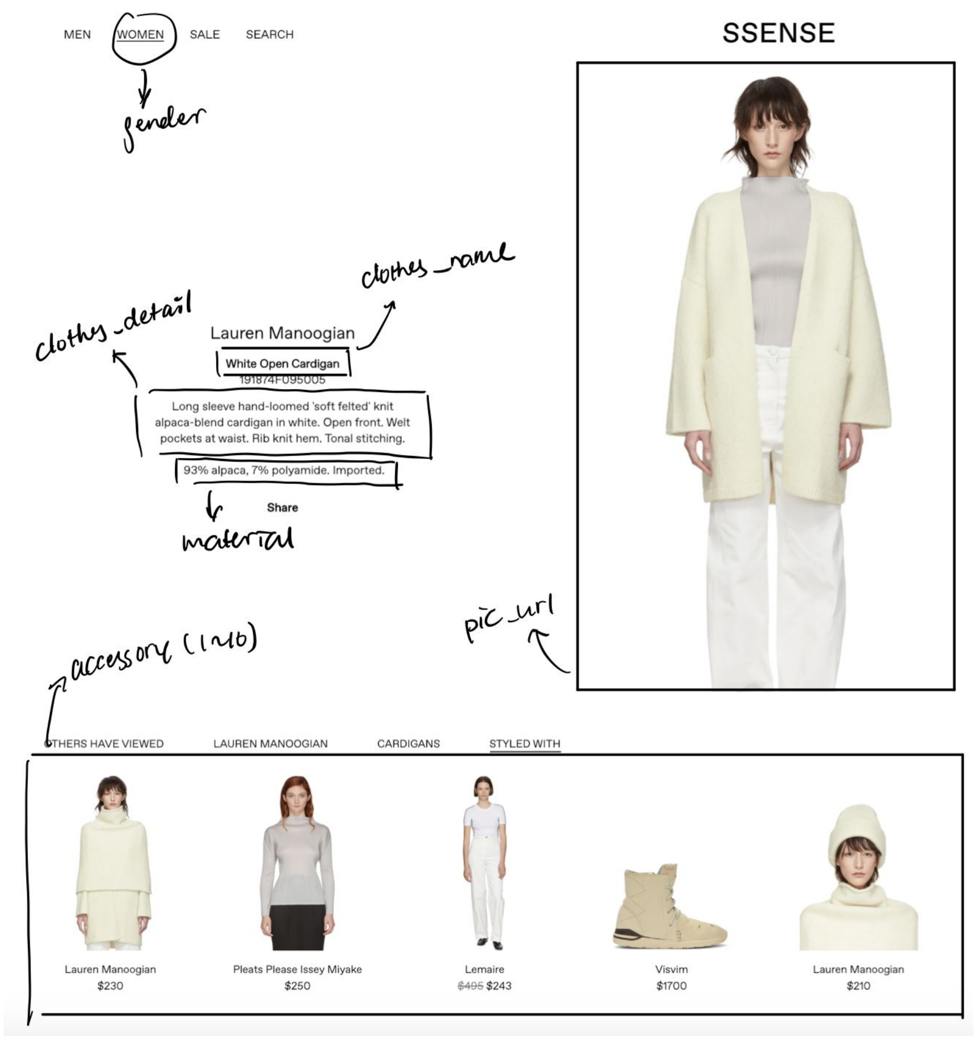
Table 1. Schemas of MenClothes, WomenClothes, MenColor, WomenColor

MenClothes / WomenClothes	MenColor / WomenColor
mcid/wcid (pk)	id (pk)
item_url	color1
clothes_name	color2
gender	color3
category	color4
pic_url	color5
clothes_detail	color6
material	color7

accessories (1-10 related items)	color8 (etc.)

In MenClothes and WomenClothes, the primary key mcid and wcid are unique integers that were initially assigned right after scraping. The item_url recorded the original url, which would be used to direct user to the ssense website. The pic_url stores the preview image of the item. Material stores the website's description of major materials used in this item. The category, clothes_detail, material, and clothes_name all contains important keywords that describes basic attributes of this particular clothes, such as color, design, material, and category. These are all fundamental attributes that we later used in FloorDog's filter function.

In MenColor and WomenColor, the primary key "id" are corresponding to the mcid and wcid. All other fields store the hexadecimal representation of 1-10 major color/colors adopted in the clothes. This data is generated by FloorDog's color recognization program as decribed above.



Long sleeve hand-loomed 'soft felted' knit alpaca-blend cardigan in white. Open front. Welt pockets at waist. Rib knit hem. Tonal stitching.

93% alpaca, 7% polyamide. Imported.

Figure 1. Examples of WomenClothes's clothes_name, gender, clothes_detail, accessory, material, and pic_url attribute.

After the database schema is drew out, we set up a local postgreSQL environment and connected it to our Django project. Through the local host, we used four postgreSQL commands to transfer all web-scraping data from csv to our database (`copy website_modelName from '/path/to/csv/file.csv' with delimiter ',' NULL as E'\'\'' CSV;`). Through the local host or QuerySet API provided by Django, we can manipulate the database according to our need.

Filter Django implementation

For the filtering function, we established four major filtering attributes based on our common experience: gender, clothing category, materials made, and primary characteristics of design. Through our scrapping program described above, we are able to collect thousands of clothing item from the ssense fashion brand. However, instead of providing keywords that directly related to filtering attributes, the website's item and material descriptions are displayed and saved into our database as long sentences.

This poses another layer of difficulty in our implementation. To solve this problem, we first sorted out possible attribute options based on word frequencies and use the QuerySet API provided by Django for database query. For color and category attributes, we further edited these options based on its meaning (merging "jacket" and "jackets"). For material and design, we choose the top group of attributes have high frequencies (more than 20 for materials and more than 230 for design).

Table 2. Examples of category, material, color, and design filtering options. The words in parenthesis represents keywords that are merged and can be searched through its related options.

Category	Material	Design	Color
"All" (default)	"All" (default)	"All" (default)	"All" (default)
blazer	cotton	Collar	beige
blouse	polyester	Long sleeve	blue
bodysuit	wool	Short sleeve	black
coat (jacket-coats)	elastane	Skim-fit	grey
jean (jeans)	polyamide	Skinny-fit	navy
skirt (skirts)	leather	Button closure	indigo
crewneck	silk	Zip closure	white
denim	alpaca	Mid-rise	khaki

After the attribute options have been set up, each item of clothes can be retrieved according to its value. At `filter_page.html`, five forms is displayed for the user to select their intended attributes. These five forms are defined in `forms.py` and used by method `filter_page()` in `views.py`. For these five options, only gender is mandatory, while others like color, material, category, and design all have a default option (value) of "All". In `views.py`, `filter_page()` gets the selected attributes and sends out a `HttpResponseRedirect()` signal that uses the `reverse()` method to generate a custom url of the filter-result page containing the selected value ("`/filter-result/W/cardigan/All/All/white`" if user choose women, cardigan, and white).

Once these filter attributes are selected, Django turns to execute the `filter_result_page()` method. The methods in turn analyzes these attributes, and generate corresponding queries using the `objects.filter()` method provided by Django's QuerySet API. In the example of white women cardigan, values of the selected color and category filters would not equal to the default value "All", thus `filter_result_page()` produces a chained filter (`objects.filter().filter()`) query to retrieved clothing items that passes through this filter. In addition, since in many cases clothes are not consisted of only one color/material/design, `method __icontains()` is used to fuzzy search through those related sentences, increasing the success rate and usefulness of the filter function.

In side `filter_result.html`, a paginator is also used in the `filter_result_page()` method to ensure the appearance of our web-page. Also, through `clothes_detail()` method, FloorDog can retrieved the original item_url of a particular displaying item, as well as redirects the user to that url on a different tab without exiting FloorDog.

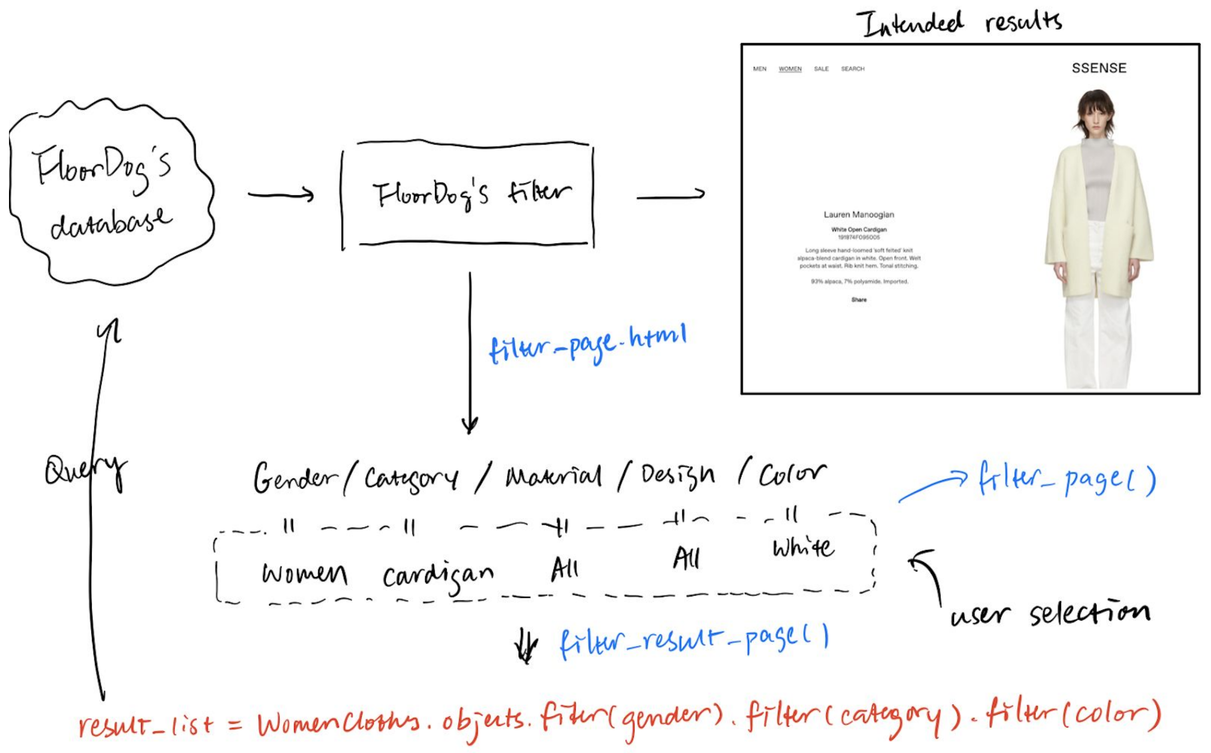

Intended results
FloorDog's database
FloorDog's filter
filter-page.html
Gender / Category / Material / Design / Color
Women cardigan All All White
filter_page()
user selection
filter_result_page()
Query
result_list = WomenCloths.objects.filter(gender).filter(category).filter(color)
MEN WOMEN SALE SEARCH
SSENSE
Lauren Manoogian
White Open Cardigan
191874F095005
Long sleeve hand-loomed 'soft felted' knit alpaca-blend cardigan in white. Open front. Welt pockets at waist. Rib knit hem. Tonal stitching.
93% alpaca, 7% polyamide. Imported.
Share

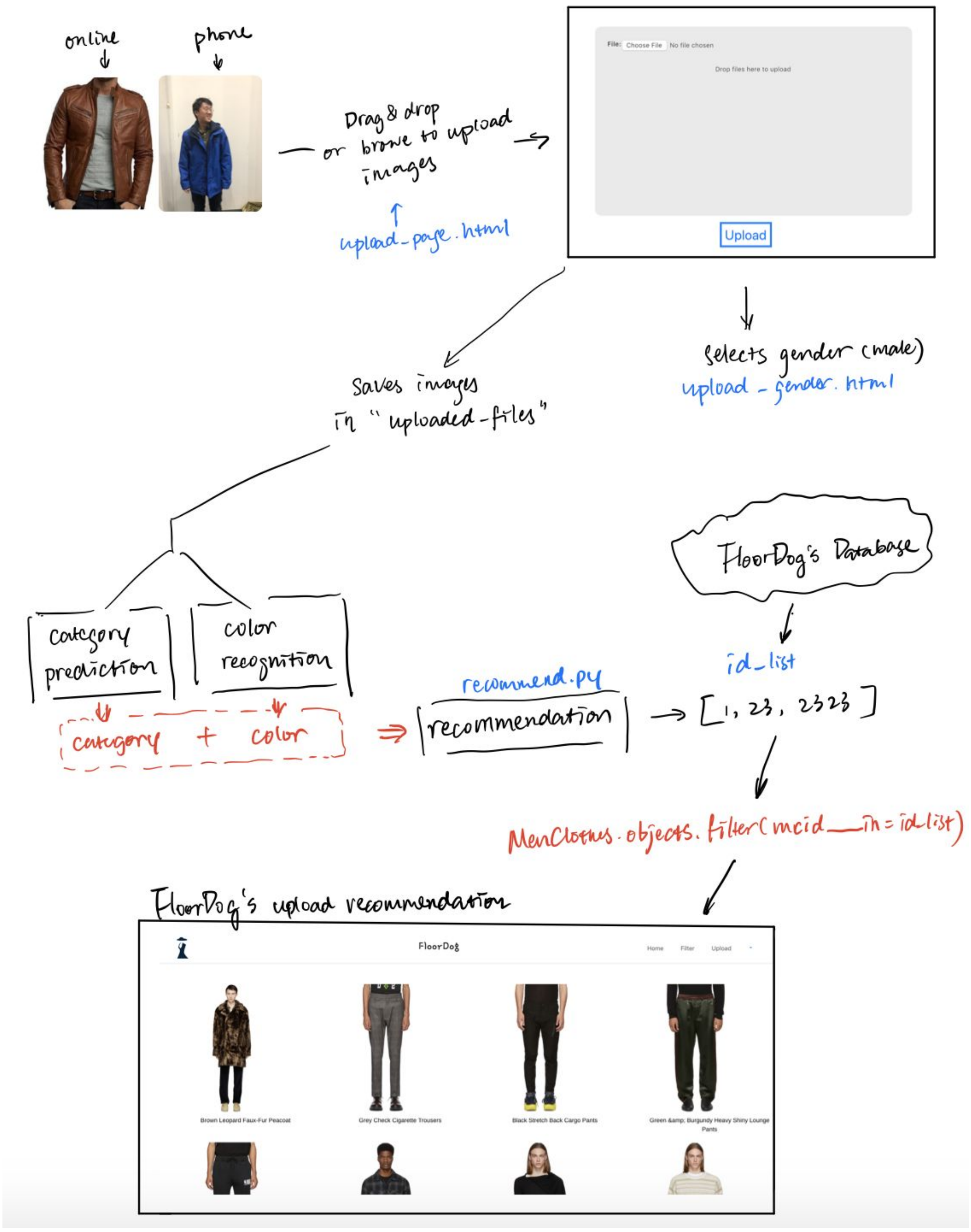
Figure 2. Sample flow chart of FloorDog's filter function if searching for a "white women cardigan".

Upload implementation

For upload recommendation, users are supposed to take a picture of his or hers already-decided outfits, upload to FloorDog under `upload_page.html`, choose his or hers gender under `upload_gender.html` and gets FloorDog's recommendation items in `upload_result.html`.

In the upload__page.html, Dropzone.js, an open source library is used to enable users upload local files through browsing as well as drag and drop. By creating a dropzone, Django sends a `UploadFileForm()` to the user, and the `upload_page()` method in view.py receives the uploaded file. Once the image is uploaded, FloorDog saved in automatically to the `uploaded_files` directory, which stored the uploaded images. The "Upload" button also directs the user to the `upload_gender.html`, where the user inputs his/hers gender in order to generate meaningful item recommendations.

The user's gender is received by the `GenderForm()`, posted by the `upload_gender_page()` method in views.py. This method later passes the user's gender to FloorDog's recommendation algorithm `rec_rec()`, which takes in the gender, recognize the clothing category of the uploaded image, and generates a list of indexes of the recommended items. This list of indexes is then process by `upload_result_page()` that generates a query (using `objects.filter()` again) to directly retrieve the recommended items based on the list of indexes. These FloorDog-recommended items are displayed similarly as `filter_result.html` in `upload_result.html`, capable of pagination and connect original website for more detail.



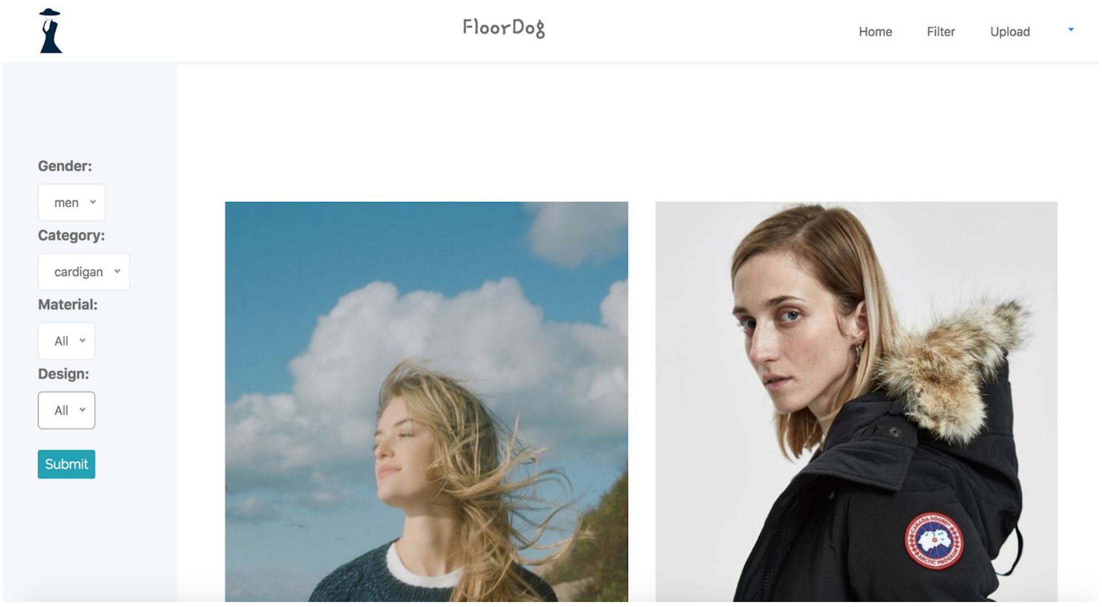
online
phone
Drag & drop or browe to upload images
upload_page.html
File: Choose File No file chosen
Drop files here to upload
Upload
Saves images in "uploaded-files"
selects gender (male)
upload - gender. html
category prediction
color recognition
category + color
recommend.py
recommendation
FloorDog's Database
id_list
[1, 23, 2323]
MenClothes.objects.filter(mcid__in=id_list)
FloorDog's upload recommendation
FloorDog
Home
Filter
Upload
Brown Leopard Faux-Fur Peacoat
Grey Check Cigarette Trousers
Black Stretch Back Cargo Pants
Green &amp; Burgundy Heavy Shiny Lounge Pants

Figure 3. Sample flow chart of FloorDog's upload function.

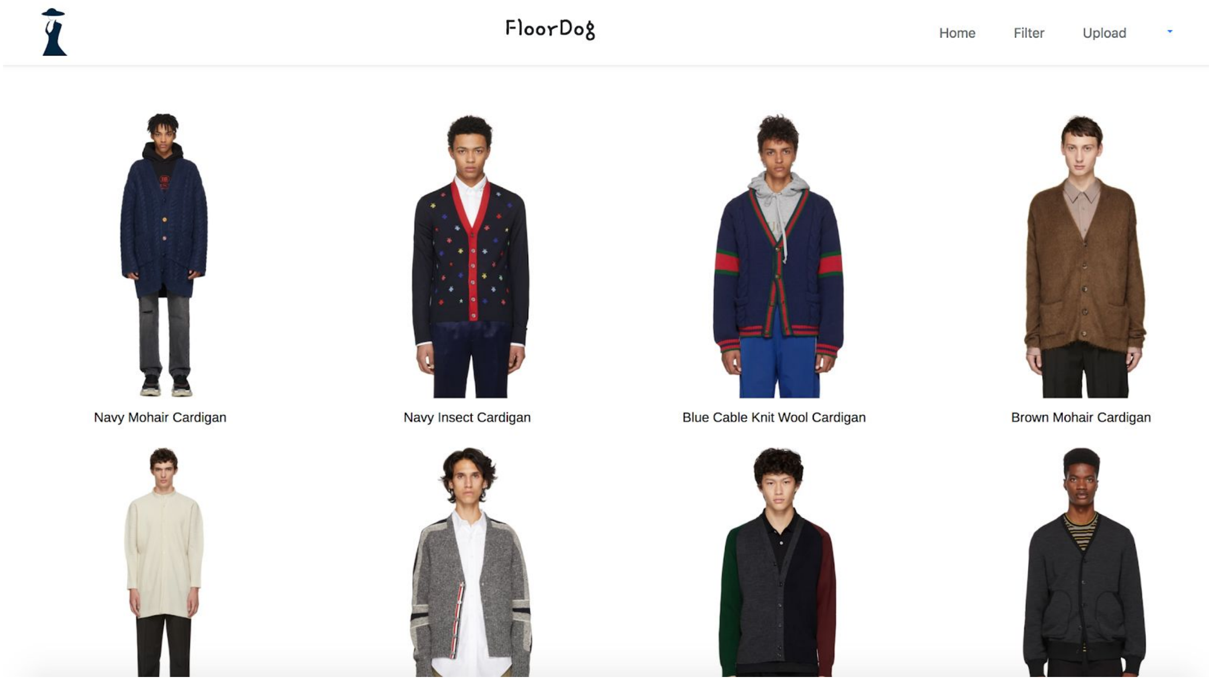
Front-end platform

Since we use Django Framework for our application, and basic structures of each page is constructed by HTML, we decided to use mainly plain HTML and CSS for the front-end platform.

We coded html and corresponding css for some shared components like banner, header and footer and reused them across different pages. Since filter_page, filter_result page and upload_result page contain multiple photos, page layout is really important for them. We used bootstrap grid to implement the layout of photos on theses pages.



External libraries like Google Fonts and Font Awesome are imported as external stylesheet to help beautify fonts via CSS and add special vector icons.



Machine Learning

For this project, we build two Deep Neural Networks based on VGG16 and Residual Net.

FashionNet

Architecture:

First, we use and improved the architecture of FashionNet proposed by *Liu, Ziwei et al*. The first half of FashionNet is the same as VGG16. The structure of VGG16 is shown in Fig. 1.

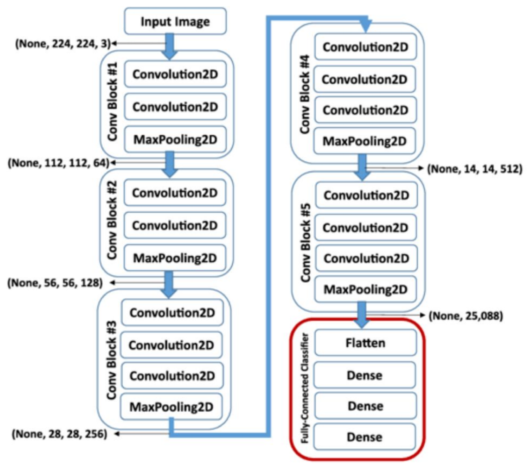


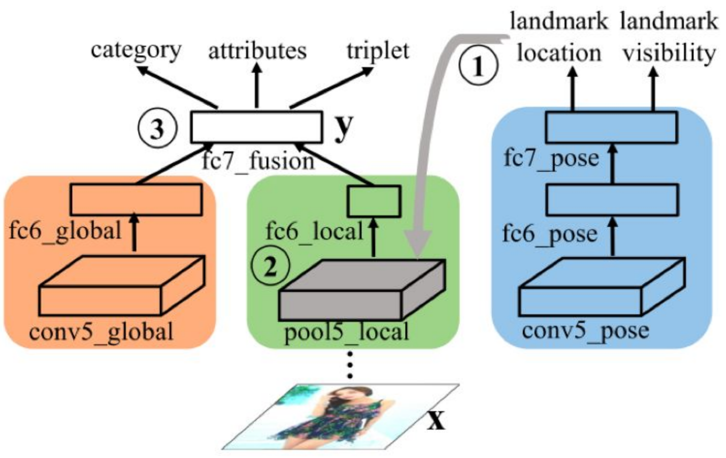
Fig. 1 Architecture of VGG16

The first four convolutional blocks are the same as in FashionNet, generating a global feature. The last convolutional block is replaced by three parts:

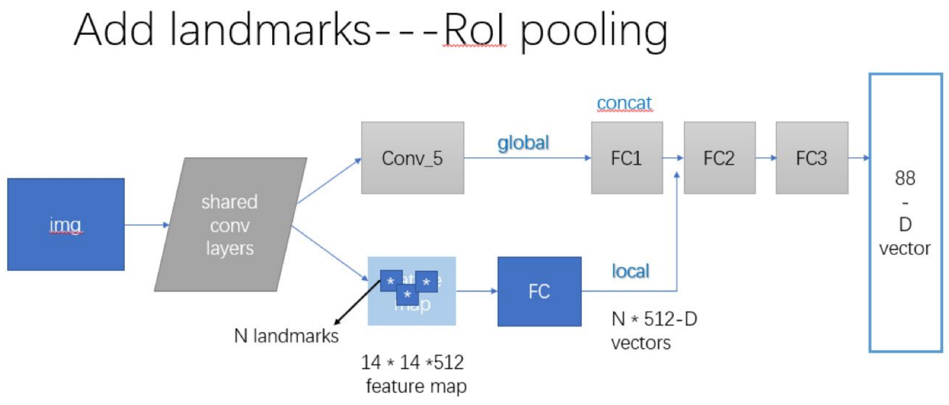
1. The first branch goes through one more convolution blocks and outputs 8 landmark locations and their visibilities (whether each landmark can be seen directly in the picture). The result is a 16*vector and a 1*8 vector
2. The second branch obtains local feature crops 8 parts from the global feature generated from the first four convolutional blocks. The result is (8-D)*7*7*512 size tensor. The tensor will go through a fully connected layer to concat with the result in branch 3.

3. The third branch continue go through one convolutional block and a fully connected layer. The result would be a 1*4096 vector

The architecture is shown in Fig. 2. This architecture is very similar to the r-cnn architecture, in which the landmarks-pooling layer is similar to the region-of-interest layer in r-cnn. Instead of doing classification after finding region of interest. FashionNet concat the local feature (obtained by branch 2) and global feature (obtained by branch 3) together to predict attributes and categories. **Note that in the original method, the net predicts attributes of a clothes, and the category is obtained by a statistical relation between category and attributes, not directly predicted by the net.** The final output would be a vector representing all attributes.



(a)



(b)
Fig. 2 FashionNet architecture

Training:

The training is based on a combined loss functions of landmark positions and attributes. The loss function for landmark positions is a linear loss (MSELoss). The loss function for attributes is a Binary Cross-Entropy loss for being a multi-label classification. The total loss will be a weighted sum of the two losses.
The training process would be a dangling procedure. For one iteration, the weight for landmark position loss is larger. For the next iteration, the weight for attributes is larger. The weight switches back and forth during the process to keep the net learning constantly from both the landmark position and attributes.

Improvements and changes:

First, we choose not to use visibility vector for landmarks. In the original fashionNet, if a landmark is not visible, we will not crop the local feature corresponding to that landmark from the global feature. Since invisible landmarks represent the spatial information of the clothes in the picture. We believe we should keep all the local feature map, even if the corresponding landmark is invisible.
Second, instead of predicting category by a statistical relationship from attributes. We extended the output by the number of categories so that the category is predicted from the neural network. The reason we make this change is that a number of attributes can hardly fix a certain category when some of the attributes might be incorrect. Even in the fashionNet paper, their result does not have a high accuracy on attributes. Therefore, the category predicted would be negatively affected.

Dataset:

We use the *DeepFashion*dataset (http://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html)published by *Multimedia Laboratory, The Chinese University of Hong Kong*. The dataset has attributes, categories and landmarks labeled for a large number of clothes.

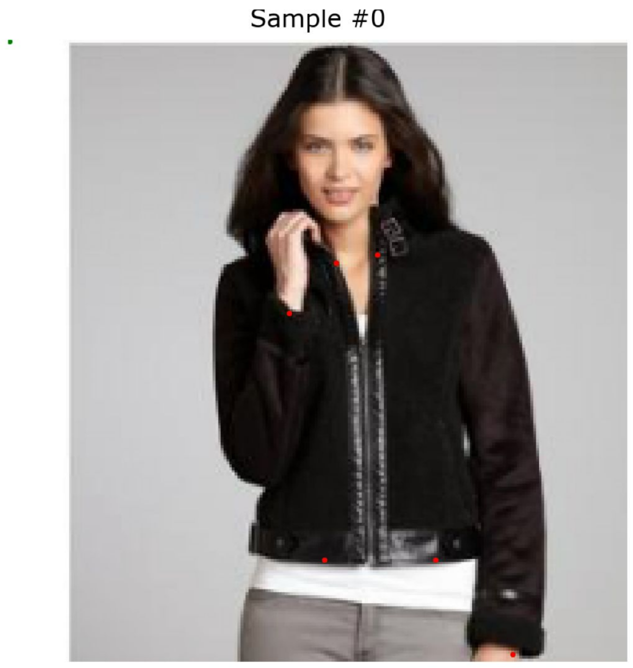


Fig. 3 A sample for landmarks(note for small red dots)

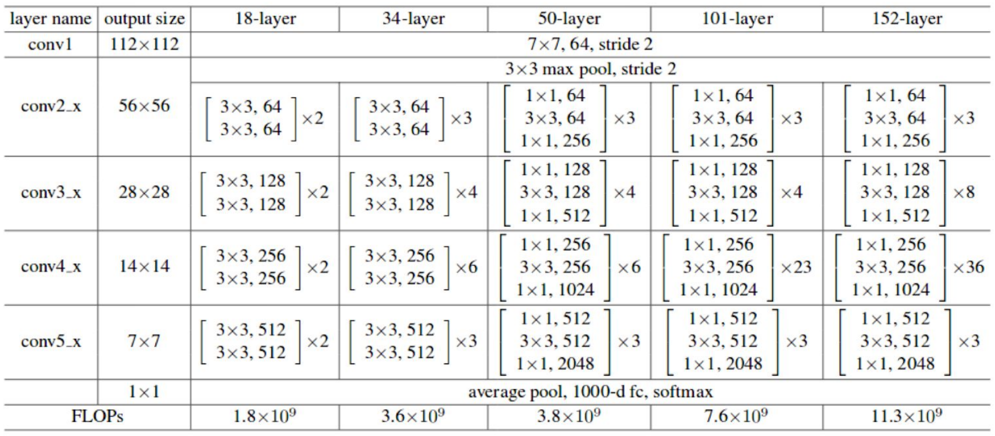
Training results:

Sadly, due to the extremely large fully connected layers in FashionNet. The forward pass and back propagate of this net is extremely slow.(more than 1 minute for a single picture) The GPU in the lab I works in is extremely crowded and I do not really get to use it for a irrelevant project. We can only conduct a training with few iterations and will not be incorporate in our final application. The result achieves about 20% accuracy on attributes, which is pretty close to the result shown by *Liu, Ziwei*in their paper. Although it seems low, it can get a pretty high accuracy on category prediction and landmark prediction.

Residual Net

Architecture:

We applied the 101 layers ResNet for the classification of clothes category. We trained two different net separately for men and women clothes, as men and women clothes has different categories.



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

Fig. 4 Residual Net architecture

Dataset:

We merged our dataset obtained from ssense.com and the DeepFashion dataset. The categories are set to be the categories we obtained from ssense.com. So I rearranged the DeepFashion dataset to fit for the new categories.

Men's clothes training data:

shorts	19270
sweaters	17566
pants	4773
jeans	7583
shirts	925
suits-blazers	7345
jackets-coats	35121
total	92583

Women's clothes training data:

shorts	19219
sweaters	17564
skirts	14698
pants	4311

dresses	70226
jeans	7511
jumpsuits	5966
jackets-coats	34824
tops	10708
total	185027

Test data are 1/30 of the whole data picked randomly.

We can see that our dataset has a biased distribution. Some categories, like men's shirts, are underrepresented. This distribution caused a lot of the false prediction in our result, which and be easily observed in the confusion matrix.

Color Recognition:

First, we can read the rgb value for every pixel in the photo. Then, the goal for color recognition is basically a clustering. Because we have only 10^4 level of points for an image, it is hard to perform some complicated clustering algorithms. Therefore, I used K-means for our program.
The problem regarding K-means is that it needs an input about how many clusters there are. However, when we take a photo for a clothing item, no one knows how many colors there will be, and we are not going to have users to manually choose that.
Therefore, considering that there cannot be too many colors, I perform a 10-cluster K-means on the photo. From the results, I calculate the distance between every two cluster center in rgb space. If I find that there are d cluster centers that are too close to other clusters (distance smaller than a threshold I set), or having too small sizes. I will then perfrom a (10-d) cluster K-means algorithm to the photo and finally gets the dominant colors in that photos.
Then from the edges and corners of the photo, I can estimate the average of background colors. If any dominant color in my result list is too close to the background color, I will eliminate it from my result.
There are a few trivial adjustments to avoid double counting when killing clusters or having nothing returned when the clothes and the background has the same color. I will not waste space here and those can all be seen in my code.

So, in summary, the process is:

Input photo -> rgb for all pixels --K-means--> 10 clusters
--filter by distance--> dominant colors
--eliminating background--> final results --> convert to hex code

Discussion of Recommendation Engine - Category & Color

In order to suggest compatible clothes, we built a recommendation engine that filtered through our database based on category and color. We wrote this code in Python and heavily utilized the pandas library.

We created broad categories for each type of clothing: top_jacket, top_casual, bottom_casual, top_formal, bottom_formal, and full_body. All labels fell under one of these broader clusters; for example, "dresses" were categorized as "full_body" and shorts under "bottom_casual". This decision let us have more data to work with when searching for compatible clothes, and helped reduce the errors made in web-scraping. Instead of having separate categories for "short" and "shorts", both fall under the broader domain of "bottom_casual".

We also wrote code that allowed us to find articles of clothing with the most similar colors. In our methodology, we first converted all hex codes into RGB values, because hex codes cannot be directly compared in a Euclidean space. For each user_input color, we searched for the item of clothing that had the most similar colors.

The algorithm for doing so involved: 1) Iterating through each row/color in the dataframe, 2) Iterating through each color given by the user, 3) Finding the minimum sum of differences for each RGB value between colors. The color that had the minimum sum of difference was the one closest to the user_input color.

We took the top five IDs of the clothes that had the smallest differences in color, and looked in our database to see the clothes that they were compatible with. These are the compatible clothes that are returned to the user.

Evalution:

It is very hard to evaluate how good our recommendation is. Since all our recommendations are based on the outfits proposed by ssensse, which is a famous fashion website, our recommendations are supposed to be good. Therefore, the performance of our application is mainly based on the performance of our machine learning module.

Also, as mentioned above, current fashion recommendation apps on the market are not targeting the same function that we are trying to do. So, there is hardly any direct comparison we can do.
Below are the results for the Residual Net module.

Accuracy:

Men' clothes	100000 iterations
overall	78%
jackets-coats	79 %
jeans	82 %
pants	78 %
shirts	64 %
shorts	80 %
suits-blazers	60 %
sweaters	81 %

Women's clothes	100000 iterations
Overall accuracy	79%
dresses	93 %
jackets-coats	82 %
jeans	86 %
jumpsuits	62 %
pants	63 %
shorts	77 %
skirts	72 %
sweaters	66 %
tops	24 %

Loss curves:

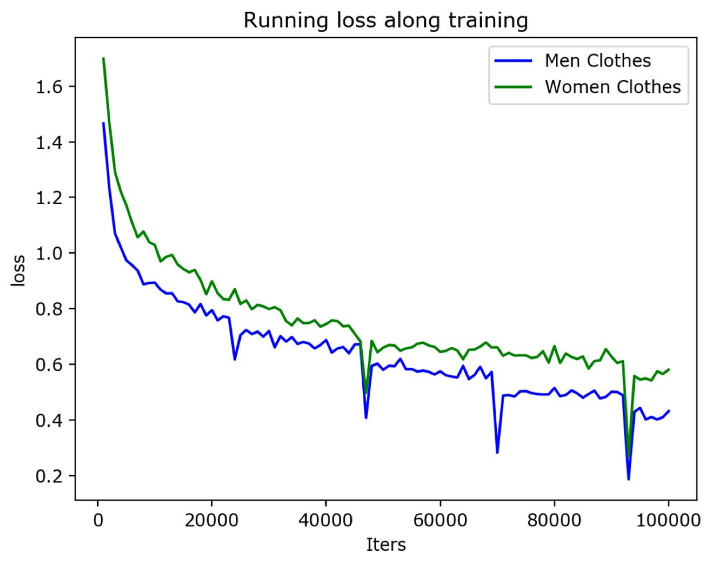
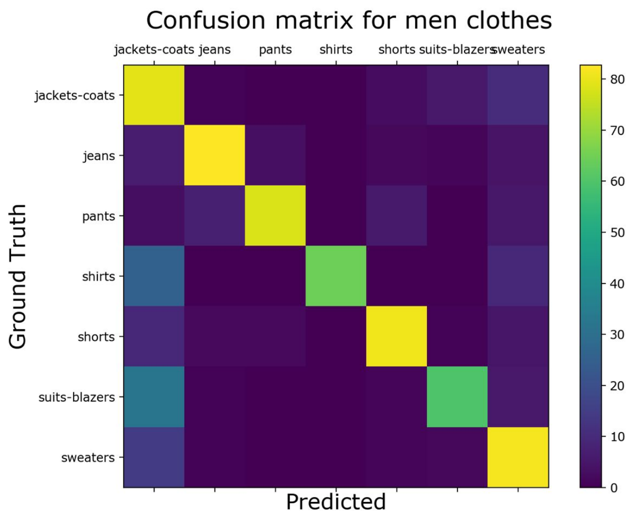


Fig. 5 Running losses for training

We can easily observe that the loss is continuing to drop along the training. This indicates that we should obtain a better performance if we have the time to keep doing the training.
From the confusion matrix below (Fig. 6), we can see that the underrepresented categories are more inclined to be predicted as other categories. This can be fixed if we can augment our dataset to balance it.

Before using this dataset, I trained a net using only the images from ssense.com. The resulting net has a 93% accuracy on test sets. But the net is very limited to the "ssense style" because *ssense* presents their clothes is a very uniformed style. Therefore we choose to abandon this dataset and net, starting to merge the *DeepFashion* dataset into our training set and testing set.



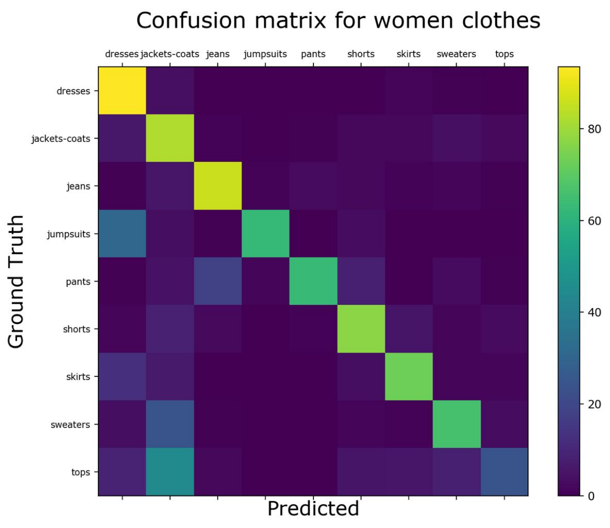
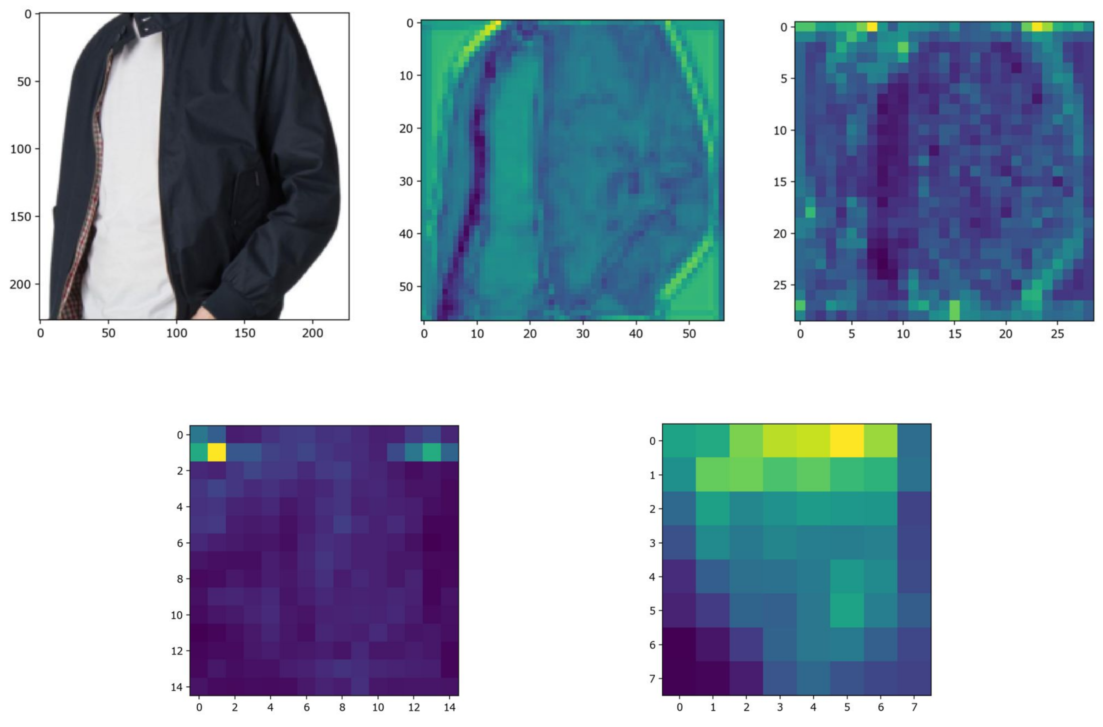
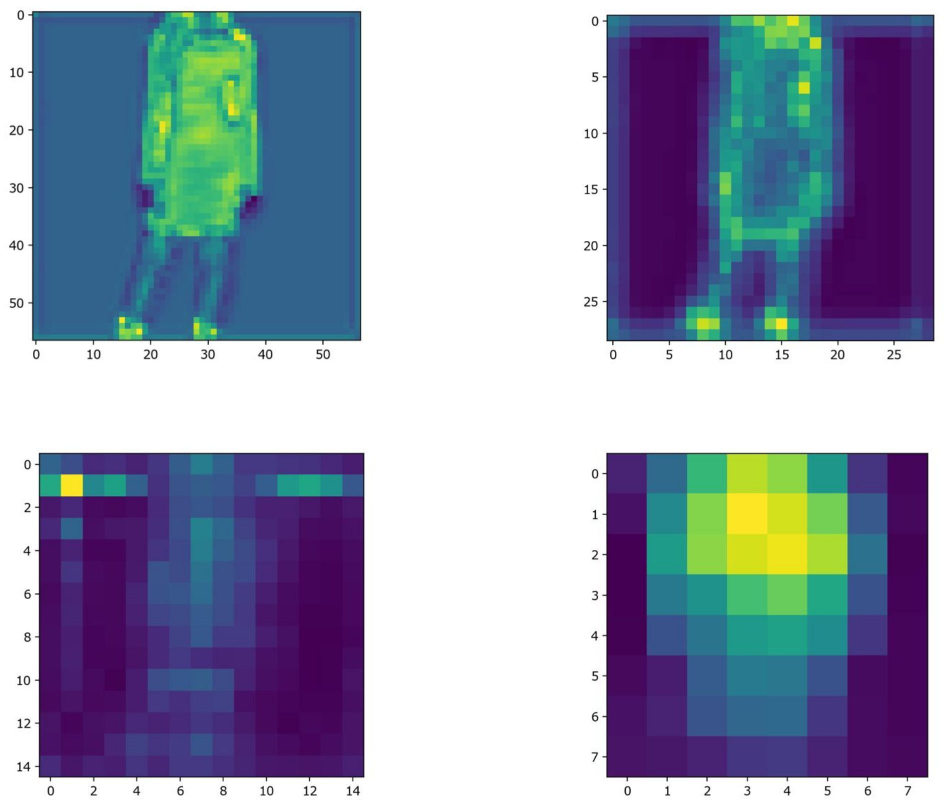


Fig. 6 Confusion matrix for ResNet 101

To better illustrate how the net works, I show the feature map in different parts of the net below in Fig. 7. This will show how the net understands and recognizes the clothes. We can clearly see that in the shallow part of the net, the net is reading some geographical feature of the clothes, like its edge, color. As the net goes deeper, it is reading more abstract information and is harder to interpret.



(a) A jacket



(b) A dress

Fig. 7 Feature maps in different stages of resnet

Furture Works

First above all, in the future we can finish the training of our improved FashionNet. After that, we can predict attributes of any input clothes, such as "long-sleeve". With predicted attributes of input clothes, we can match up the input clothes with the clothes in our database more accurately, and consequently generate more accurate recommendation.

Also with the attribute predicting module, we can extend our database to clothes selling by other shops, like Uniqlo, that they do not have detailed description about each clothes on their websites.

Moreover, for neural network training, I have trained a CNN module that only predicts landmark position and visibilities. Using this module, we can easily add any clothes picture to our training set without having to manually label the landmarks.

In the broader future, we would like to have users to upload their own sets of outfits. We can do category and attributes prediction for each single clothing in the sets to fit in our database. Meanwhile, we will record the compatible relationships between those clothing items and will be used to recommend outfits for other users. In this way, our application will be growing with users and actually adapt to the current fashion trend.