Jessica Fogerty
12/1/19
Final Project
Machine Learning II

FocusedFashion
Individual Report

Introduction

We all have our favorite pieces of clothing that we consistently wear. Over time, these clothing items may not fit anymore or degrade in quality. Subsequently, it may be difficult to find the same clothing item or time-intensive to find similar clothing items due to the vast amount of clothing websites and clothing retail stores. For example, the clothing retailer H&M has hundreds of fashion options available on their website and a subpar filtering system, making it difficult to find clothing pieces of interest. This is a common issue for both of us, and it is something we attempted to streamline by using deep learning techniques. Our solution involves a two-step approach. First, we trained convolutional neural networks (CNN) on fashion images in order to extract the feature maps associated with different clothing items. Second, we used those feature maps as inputs to a k-nearest neighbors (KNN) model that found the five closest neighbors to a given query image that served as recommendations.

Prior research has found that utilizing deep learning techniques in fashion recommendation systems has proven to be better than traditional recommendation techniques. Le (2019) trained a ResNet neural network on tops to classify these clothing images into six classes and then obtain feature maps. Then, the author implemented a nearest-neighbor based search on the feature maps. Similarly, Tuinhof, Pirker, & Haltmeier (2018) trained fashion images using AlexNet and BN-inception to extract feature maps. They used those feature maps to implement a KNN to return ranked recommendations. Our solution is similar to the previous literature except that we experimented with different network architectures and KNN implementations/distance metrics in the hopes of achieving efficient and superior fashion recommendations.

The remainder of the paper is structured as follows. The Individual Work section outlines the work I completed individually for my project, it also includes work that both Mary and I collaborated together on. Contribution provides a detailed description of my individual contribution to the project. Results provides a numerical analysis of my contributions. This paper concludes with a short discussion in the Summary section. The code section provides a very

rough estimate of the amount of code I wrote as a ratio of the total amount of code that I contributed.

Contribution

Throughout this project both Mary and I contributed equally in terms of workload. There were several pieces of this project that we worked on together and then several pieces that we worked on individually. Both Mary and I worked on the script titled 'baseline_model_template.py'. This was the initial model that we built, each subsequent model was a variation of this script, it was a good starting place for the modeling portion of our project. This script also includes the class that creates our dataset (in preparation of model training), the data loader function and the training for- loop. The next script we both worked on together is titled 'download_fashion_dataset.py' Nlecoy (2018). This script loops through the JSON file and reads in the image id and the associated image via a url that was provided. The last script we both worked on is titled 'scrape_banana_republic_images.py'. This script was used to scrape the Banana Republic images that we used in our recommendation system. In regards to the group report, presentation, proposal and README, we both collaborated and or split the work equally.

As mentioned above, in addition to the pieces of the project Mary and I worked on together there were several pieces that we worked on individually. I worked on the script titled 'obtain_fashion_dataset_labels.py' (based on Nlecoy (2018)). This script loops through the JSON file and loads the labels that are associated with the correct images. These image ids and labels were then placed into a pandas dataframe and saved for later use. Next I worked on a script called 'helper_functions.py'. These functions were helpful in helping us figure out what the smallest image size was in the dataset. The second function helped us determine if the training, validation and test set had the same number of labels. The next scripts I wrote were the six simple cnn model scripts titled 'jessica_model_1.py' through 'jessica_model_7.py'. These scripts are similar to the baseline model template except the model architecture and the hyperparameters vary across scripts. In addition to writing the aforementioned scripts I was responsible for exploratory data analysis that was performed on the labels and creating the necessary visualizations. I also created many of the visualizations used in the presentation. Those pieces of the project can be seen in the script titled 'presentation_visualizations.py'.

Individual Work

The first portion of my individual work that I will discuss is how the labels were handled. First a script was written to load the labels (Figure 1). As mentioned above this code loads the labels from the JSON file and creates pandas dataframes which consists of the image ID and its corresponding label for each train, test and validation data sets. I can also be seen how we subset the data by index in the 'create test with labels' and the 'only keep first 300000 rows' sections of the code snippet. This code was adapted from Nlecoy (2018).

```
train={}
test={}
validation={}
with open('%s/train.json'%(data_path)) as json_data:
    train= json.load(json_data)
with open('%s/test.json'%(data_path)) as json_data:
    test= json.load(json_data)
with open('%s/validation.json'%(data_path)) as json_data:
    validation = json.load(json_data)

print('Train No. of images: %d'%(len(train['images'])))
print('Test No. of images: %d'%(len(test['images'])))
print('Validation No. of images: %d'%(len(validation['images'])))

#Train
train_img_url=train['images']
train_img_url=pd.DataFrame(train_img_url)
train_ann=train['annotations']
train_ann=pd.DataFrame(train_ann)
train=pd.merge(train_img_url, train_ann, on='imageId', how='inner')

#Test
test=pd.DataFrame(test['images'])

#Validation
val_img_url=validation['images']
val_img_url=pd.DataFrame(val_img_url)
val_ann=validation['annotations']
val_ann=pd.DataFrame(val_ann)
validation=pd.merge(val_img_url, val_ann, on='imageId', how='inner')

#create test with labels
test2 = train_ann[700000:730000]


#only keep first 300000 rows
train_ann = train_ann[:300000]
```

Figure 1. Load labels code snippet

In addition to loading the data I performed an exploratory data analysis. The first step in that analysis was to ensure that each train, test and validation sets included the same number of labels. For example, early on in the project we found that the test set was missing an observation including a minority label (Figure 2 shows a code snippet), the function checks that each of the datasets have the same number of labels present.

```
def cross_ref_Labels(data_path, image_path):
    images = []
    images += [each for each in os.listdir(image_path)]
    images.sort()
    df = pd.read_csv(data_path)
    df['file name'] = df['imageId'].apply(lambda x: '{}.jpg'.format(x))
    mask = df['file name'].isin(images)
    return print(mask.value_counts())
```

Figure 2. Check label counts.

Next I further explored the labels by looking at the number of labels per image. I used the csv that was saved out after creating the labels and created a new column that included the length of the list of the labels associated with each image. Next a histogram was created that illustrates the distribution of the number of labels per image (Figure 3). The average number of labels per image is 36, while the maximum is 142 and the minimum is 6. Then, I created a word cloud in order to evaluate the frequency of the labels within the training set (Figure 4). In addition to the histogram I used the label_map csv (Visipedia, 2018), this csv included the Label number and its corresponding attribute. Then, I created a word cloud in order to evaluate the frequency of the labels within the training set (Figure 4). The size of the label text indicates the frequency of the label. From this word cloud, it is apparent that the labels female, long sleeved, black, round neck, and sleeveless appear to be the most prevalent labels, while purple, tank tops, and t-shirts appear to be less prevalent labels. Ultimately, this word cloud indicates that there is a label imbalance within the training set that may potentially influence our fashion recommendations.

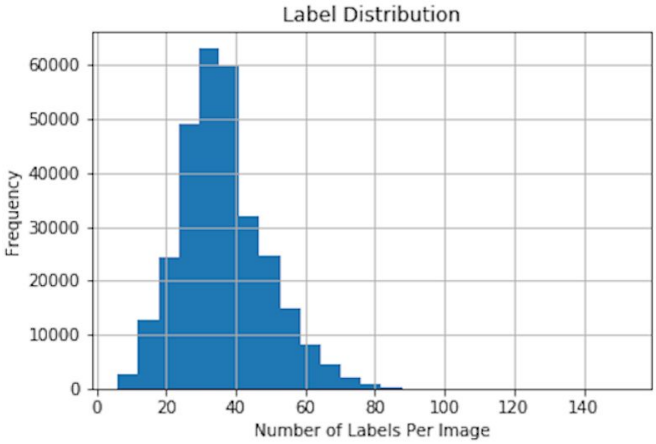


Figure 3. The distribution of labels per image within the training set.

Figure 3. A word cloud, highlighting the frequency of the labels within the training set.

In addition to examining the labels it was important for us to determine the size of the smallest image in the dataset. We needed to know this in order to determine what size image we should start with when transforming the data for modeling.

In order to obtain the best model, I built multiple simple cnn models. Each of the simple cnn models vary in terms of architecture and hyperparameters (Figure 4a-b). Of note, Jessica 2 was removed due to a corrupted model file. Overall, we tried a variety of different hyperparameter combinations (Figure 4a-b). I decided to experiment with the learning rate by increasing and decreasing it in pursuit of better results. I also experimented with a smaller dropout rate. I found that the smaller dropout rate was more advantageous in this case for obtaining a higher micro-averaged F1-score. Also, Mary and I agreed that due to the nature of fashion recommendation, overfitting may not be a high-priority issue as it is in classification tasks. I determined that increasing the number of epochs was beneficial in obtaining a higher micro-averaged F1-score. Each of the simple CNN models utilize the Adam optimizer, which was chosen based on the previous literature regarding deep learning-based fashion recommendation (Tuinhof, Pirker, & Haltmeier, 2018). Experimenting with the batch size proved to be difficult due to computational constraints. In this case, the smallest batch size was 200 images and the largest batch size was 280 images. Besides adjusting hyperparameters, I also experimented with resizing the images ranging from sizes of 32x32 to 120x120 and performing a variety of image transformations (Figures 5-6).

Figure 4. Best simple CNN Model

HyperParameters	**Model: Baseline**	**Model: Baseline [Jessica 1]**	**Model: Baseline [Jessica 3]**	**Model: Baseline [Jessica 4]**	**Model: Baseline [Jessica 5]**
Learning Rate	1.00E-02	5.00E-03	5.00E-05	5.00E-07	5.00E-05
Dropout	0.5	0.1	0.1	0.005	0.01
Epochs	5	5	10	10	25
Optimizer	Adam	Adam	Adam	Adam	Adam
Loss	BCEWithLogitsLoss ()	BCEWithLogitsLoss ()	BCEWithLogitsLoss ()	BCEWithLogitsLoss ()	BCEWithLogitsLoss ()
Activation function	Relu	Relu	Relu	Relu	Relu
Batch Size	256	256	200	280	200
Layers	4 (2 conv, 2 linear)	4 (2 conv, 2 linear)	4 (2 conv, 2 linear)	4 (2 conv, 2 linear)	4 (2 conv, 2 linear)

Figure 4a. A table of hyperparameters of each model

HyperParameters	Model: Baseline [Jessica 6]	Model: Baseline [Jessica 7]		
Learning Rate	5.00E-05	5.00E-05		
Dropout	0.01	0.01		
Epochs	50	25		
Optimizer	Adam	Adam		
Loss	BCEWithLogitsLoss()	BCEWithLogitsLoss()		
Activation function	Relu	Relu		
Batch Size	200	200		
Layers	4 (2 conv, 2 linear)	4 (2 conv, 2 linear)		

Figure 4b. A table of hyperparameters of each model

```
def create_data_loader(img_dir, info_csv_path, batch_size):
    """Returns a data loader for the model."""
    img_transform = transforms.Compose([transforms.Resize((32, 32), interpolation=Image.BICUBIC),
                                        transforms.ToTensor()])
    img_dataset = FashionDataset(img_dir, img_transform, info_csv_path)
    data_loader = DataLoader(img_dataset, batch_size=batch_size, shuffle=True, num_workers=12, pin_memory=True)
    return data_loader
```

Figure 5. Jessica Model 1

```
def create_data_loader(img_dir, info_csv_path, batch_size):
    """Returns a data loader for the model."""
    img_transform = transforms.Compose([transforms.Resize((50, 50), interpolation=Image.BICUBIC),
                                        transforms.ToTensor()])
    img_dataset = FashionDataset(img_dir, img_transform, info_csv_path)
    data_loader = DataLoader(img_dataset, batch_size=batch_size, shuffle=True, num_workers=12, pin_memory=True)
    return data_loader
```

Figure 6. Jessica Models 3-7

Ultimately, the best simple CNN model was Jessica 5 (Figure 4). It consists of two convolutional layers using a kernel size of 3, a stride of 1, and a padding of 1. After each convolutional layer, there is a batch normalization layer and max pooling layer that utilizes a kernel size of 2 and a stride of 2. Then, the two convolutional layers are followed by a linear layer with a relu activation function, batch normalization layer, and a dropout layer. The output layer is a linear layer that outputs a 1-D array of size 149, which is the total number of unique labels (Figure 5). The total number of parameters that this simple CNN model includes is 2,642,851 (Figure 6).

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 35, (3, 3), stride=1, padding=1)
        self.convnorm1 = nn.BatchNorm2d(35)
        self.pool1 = nn.MaxPool2d((2, 2), stride=2)

        self.conv2 = nn.Conv2d(35, 70, (3, 3), stride=1, padding=1)
        self.convnorm2 = nn.BatchNorm2d(70)
        self.pool2 = nn.MaxPool2d(kernel_size=(2, 2), stride=2)

        self.linear1 = nn.Linear(10080, 256)
        self.linear1_bn = nn.BatchNorm1d(256)
        self.drop = nn.Dropout(DROPOUT)
        self.linear2 = nn.Linear(256, 149)

        self.act = torch.relu

    def forward(self, x):
        x = self.pool1(self.convnorm1(self.act(self.conv1(x))))
        x = self.pool2(self.convnorm2(self.act(self.conv2(x))))
        x = self.drop(self.linear1_bn(self.act(self.linear1(x.view(len(x), -1)))))
        x = self.linear2(x)
        return x
```

Figure 4. Best simple CNN model.

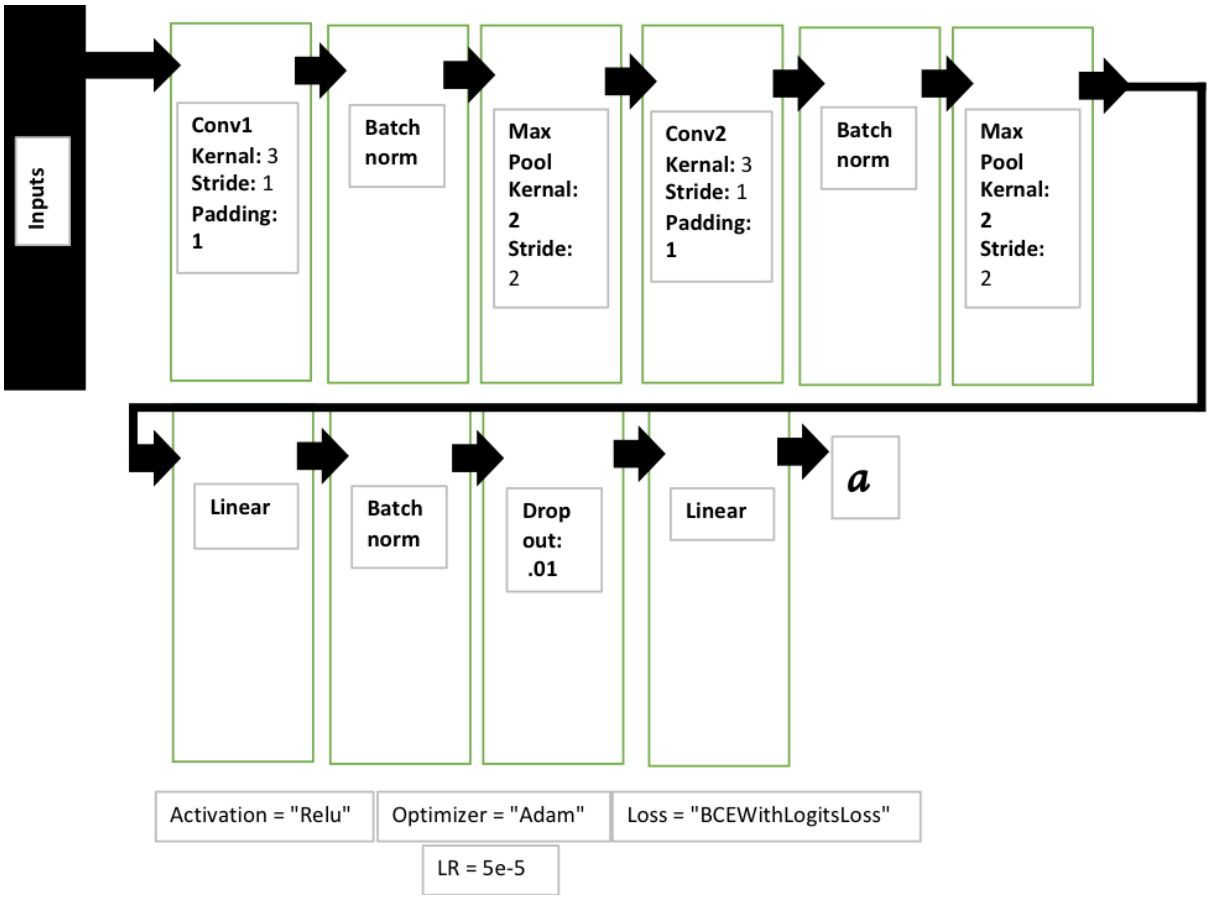
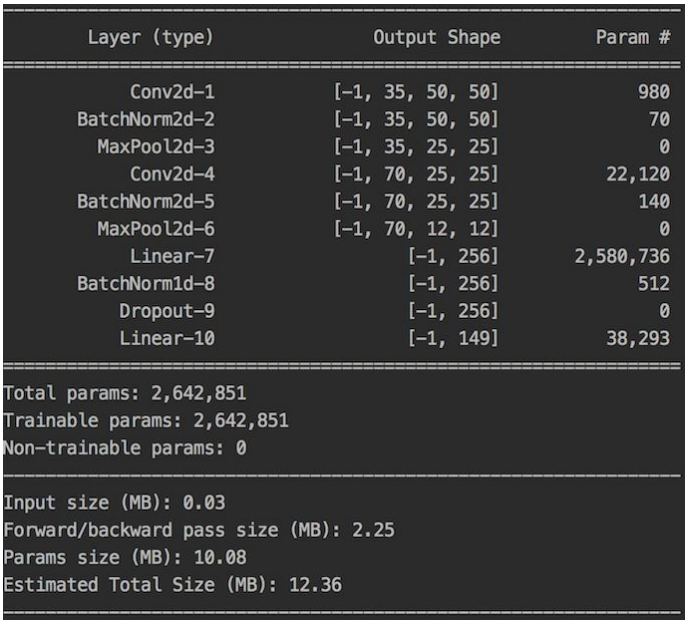


Figure 7. The architecture of the best simple cnn.



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 35, 50, 50]	980
BatchNorm2d-2	[-1, 35, 50, 50]	70
MaxPool2d-3	[-1, 35, 25, 25]	0
Conv2d-4	[-1, 70, 25, 25]	22,120
BatchNorm2d-5	[-1, 70, 25, 25]	140
MaxPool2d-6	[-1, 70, 12, 12]	0
Linear-7	[-1, 256]	2,580,736
BatchNorm1d-8	[-1, 256]	512
Dropout-9	[-1, 256]	0
Linear-10	[-1, 149]	38,293

Total params: 2,642,851
Trainable params: 2,642,851
Non-trainable params: 0

Input size (MB): 0.03
Forward/backward pass size (MB): 2.25
Params size (MB): 10.08
Estimated Total Size (MB): 12.36

Figure 6. The trainable parameters and output sizes of the best simple cnn.

In addition to training these models I utilized the training text files and the test text files to create visualizations of the training loss and distribution of the F1 scores in order to assist in analyzing the models performance. In order to do this I had to write a function to plot both the loss and F1 metrics but also a function that takes a text file as input and returns a clean file that can be used for plotting. Figure 7 illustrates this function.

```
def clean_txt_file(txt_file, new_txt_file):
    """Use this to clean up the .txt files for use
    Removes lines containing strings we wish to remove and outputs to new txt file.
    Load the new txt file to plot the data"""
    words_to_remove = ['MODEL_NAME', 'EPOCH', 'Validation']
    with open(txt_file) as oldfile, open(new_txt_file, 'w') as newfile:
        for line in oldfile:
            if not any(words_to_remove in line for words_to_remove in words_to_remove):
                newfile.write(line)
```

Figure 7. The text file cleaning function

Results

Overall, the best simple CNN model was Jessica 5. During training, as the number of batches increased, the training the loss decreased. The tail of the training loss plot appears to continually decrease, indicating that the training loss may still continue to decrease with more batches and could potentially increase model performance (Figure 8). As we had compute power limitations we used a batch size of 64 inorder to test the models. On the test set, the average micro-averaged

F1-score was 0.390, the maximum was 0.410 and the minimum was 0.360 (Figure 9). I was particularly proud of this model as its micro-averaged average F1 score was only .06 points lower than the MobileNet model.

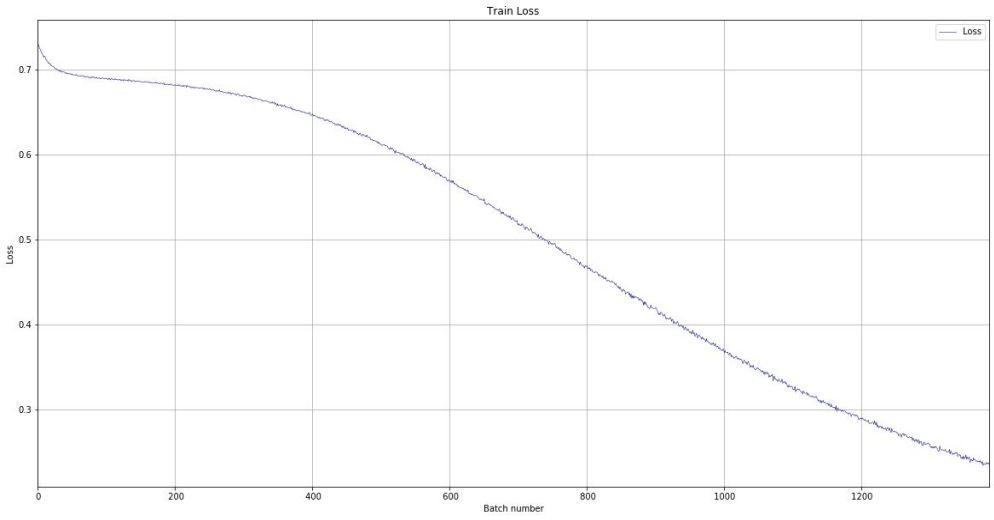


Figure 8. The training loss for the best simple cnn model..

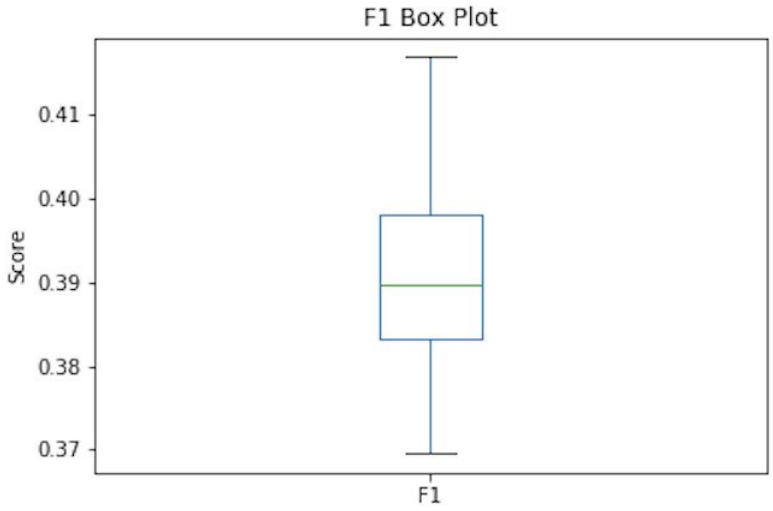


Figure 10. Micro-averaged F1-score distribution for the best simple cnn model.

Summary & Conclusions

Throughout this project Mary and I presented a clothing recommendation system that utilizes deep learning techniques in order to obtain more accurate recommendations than the trivial recommendation system. Both our joint work and individual work was crucial in reaching our conclusion. My specific work was able to provide clear information about the labels in the

dataset. Information that helped us ensure that each train, test and validation set had the same number of labels. The label exploratory data analysis also helped us understand how the label imbalance could have affected the recommendations that our best KNN model produced. In addition to providing and exploratory data analysis on the labels my work was able to identify the best simple cnn model that we trained, the model titled 'Baseline [jessica 5]'. The hyperparameters that seemed to make a difference were: the learning rate, number of epochs and drop out. This model did not perform as well as MobileNet (the model Mary used) as it received an average F1 score of .06 less and the recommendations in our opinion were not as accurate. Regardless, I was extremely proud that it performed the way that it did.

I learned a great deal throughout this project. I learned how to use JSON files (this was my first time using such a file) and I learned how to solve a multi label image classification problem. Additionally, I learned a great deal about CNN architecture and how to change it and the hyperparameters in order to build a better model. I also learned that CNN can be used for more than just image classification. We were able to utilize the CNN model in a creative way by extracting the feature maps from it. I also learned that keeping organized and communicating well with your project partner is really important, it also helps to have a good project partner.

As for what I would improve in the future, I would certainly start by addressing the compute power limitation problem. I would also utilize the DeepFashion and DeepFashion2 datasets. Both include bounding boxes and pose estimations that we can use to extract feature maps for individual clothing items. The recommendations that were produced make it clear that bounding boxes would be crucial for recommendation improvement and that the label imbalance problem needs to be addressed. In addition to addressing all these problems both Mary and I want to continue with this project and create a mobile/web application for others to use.

Code

It is really hard to provide an accurate estimate, we have a lot of code and we wrote a fair amount of it together. I did utilize code from the internet but I also wrote a fair amount of code from scratch. I would feel comfortable saying that out of approximately 580 lines of code (a combination of my own code and code mary and I collaborated on) I borrowed and used 50%.

References

Anqitu. (2018, April 11). For Starter: JSON to MultiLabel in 24 seconds. Retrieved from https://www.kaggle.com/anqitu/for-starter-json-to-multilabel-in-24-seconds.

Aspris, M. (2018, September 15). Simple Implementation of Densely Connected Convolutional Networks in PyTorch. Retrieved from https://towardsdatascience.com/simple-implementation-of-densely-connected-convolutional-networks-in-pytorch-3846978f2f36.

Gómez, Raúl. (2018). Retrieved from https://gombru.github.io/2018/05/23/cross_entropy_loss/.
Guérin, J., & Boots, B. (2018). Improving Image Clustering With Multiple Pretrained CNN Feature Extractors, 1–10.
He, T., & Hu, Y. (2018). FashionNet: Personalized Outfit Recommendation with Deep Neural Network, 1–8.
Hollemans, M. (2017). Retrieved from https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/.
Hollemans, M. (2018). Retrieved from https://machinethink.net/blog/mobilenet-v2/.
Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., … Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.
Huang, G., Liu, Z., & van der Maaten, L. (2018). Densely Connected Convolutional Networks. Densely Connected Convolutional Networks, 1–8. Retrieved from https://arxiv.org/pdf/1608.06993.pdf
iMaterialist Challenge (Fashion) at FGVC5. (2018). Retrieved from https://www.kaggle.com/c/imaterialist-challenge-fashion-2018.
Le, J. (2019, August 16). Recommending Similar Fashion Images with Deep Learning. Retrieved from https://blog.floydhub.com/similar-fashion-images/.
Mratsim. (2017, October 4). Starting Kit for PyTorch Deep Learning. Retrieved from https://www.kaggle.com/mratsim/starting-kit-for-pytorch-deep-learning.
Nazi, Z. A., & Abir, T. A. (2018). Automatic Skin Lesion Segmentation and Melanoma Detection: Transfer Learning approach with U-Net and DCNN-SVM. International Joint Conference on Computational Intelligence. Retrieved from https://www.researchgate.net/publication/330564744_Automatic_Skin_Lesion_Segmentation_and_Melanoma_Detection_Transfer_Learning_approach_with_U-Net_and_DCNN-SVM
Nlecoy. (2018, April 4). iMaterialist downloader util. Retrieved from https://www.kaggle.com/nlecoy/imaterialist-downloader-util.
PyTorch. (2017, December 12). DenseNet-161. Retrieved from https://www.kaggle.com/pytorch/densenet161.
Renatobmlr. (2018, November 11). [PyTorch] DenseNet as Feature Extractor. Retrieved from https://www.kaggle.com/renatobmlr/pytorch-densenet-as-feature-extractor.
sklearn.metrics.f1_score. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.
Sharma, P. (2019, August 1). Build your First Multi-Label Image Classification Model in Python. Retrieved from https://www.analyticsvidhya.com/blog/2019/04/build-first-multi-label-image-classification-model-python/.
Shrimali, V. (2019, June 3). Home. Retrieved from https://www.learnopencv.com/pytorch-for-beginners-image-classification-using-pre-trained-models/.
sklearn.neighbors.BallTree. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html#sklearn.neighbors.BallTree.
Spotify. (2019, October 24). spotify/annoy. Retrieved from https://github.com/spotify/annoy.

Tsang, S.-H. (2019, March 20). Review: DenseNet - Dense Convolutional Network (Image Classification). Retrieved from https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803.
Tuinhof, H., Pirker, C., & Haltmeier, M. (2018). Image Based Fashion Product Recommendation with Deep Learning, 1–10.doi.org/10.1007/978-3-030-13709-0_40
Utkuozbulak. (2018, July 15). utkuozbulak/pytorch-custom-dataset-examples. Retrieved from https://github.com/utkuozbulak/pytorch-custom-dataset-examples.
Visipedia. (2019, June 26). visipedia/imat_fashion_comp. Retrieved from https://github.com/visipedia/imat_fashion_comp.