FocusedFashion Individual Report
Mary Gibbs

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 Contributions section provides a description of my contributions to the project. The Individual Work section contains the work that I completed individually as well as the work that both Jessica and I collaborated together on for this project. The Results and Summary & Conclusions sections provides a description of the results and summary & conclusions pertaining to my contributions to the project. The Code section includes a calculation of the code I found or copied from the internet.

Contributions

I worked on these Python scripts: mary_model_1.py, mary_model_2.py, densenet_model.py, mobilenet_model.py, test_models.py, baseline_recommendations.py, and generate_recommendations.py. Jessica and I worked on these Python scripts together: download_fashion_dataset.py based on Nlecoy (2018), baseline_model_template.py, and scrape_banana_republic_images.py. In terms of figures, I created the dataset preprocessing

diagram, recommendation system diagram, and all of the recommendations visualizations. In terms of the group proposal, README, presentation, and group report, Jessica and I either worked together or evenly split the writing associated with these requirements.

Individual Work

The first step in our approach involves creating and training CNNs that will be utilized for feature extraction and subsequent KNN ranking recommendations. Therefore, based off of our baseline_model_template.py, I created the simple CNNs Mary 1 and Mary 2 found in mary_model_1.py and mary_model_2.py, respectively. I decided to experiment with various image sizes and larger kernel sizes (Figure 1a-b).

```
def create_data_loader(img_dir, info_csv_path, batch_size):
    """Returns a data loader for the model."""
    img_transform = transforms.Compose([transforms.Resize((100, 100), 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
```

```
MODEL_NAME = "mary_model_1"
LR = 0.1
N_EPOCHS = 5
BATCH_SIZE = 1024
DROPOUT = 0.45


class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, (12, 12), stride=2, padding=1)
        self.convnorm1 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d((2, 2), stride=2)

        self.conv2 = nn.Conv2d(32, 64, (6, 6), stride=2, padding=1)
        self.convnorm2 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d(kernel_size=(2, 2), stride=2)

        self.linear1 = nn.Linear(64*5*5, 1024)
        self.linear1_bn = nn.BatchNorm1d(1024)
        self.drop = nn.Dropout(DROPOUT)
        self.linear2 = nn.Linear(1024, 149)

        self.relu = torch.relu

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

Figure 1a. The data loader and simple CNN Mary 1 code.

```
def create_data_loader(img_dir, info_csv_path, batch_size):
    """Returns a data loader for the model."""
    img_transform = transforms.Compose([transforms.Resize((120, 120), 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
```

```
MODEL_NAME = "mary_model_2"
LR = 0.01
N_EPOCHS = 10
BATCH_SIZE = 1024
DROPOUT = 0.50


class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, (12, 12), stride=2, padding=1)
        self.convnorm1 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d((2, 2), stride=2)

        self.conv2 = nn.Conv2d(32, 64, (8, 8), stride=2, padding=1)
        self.convnorm2 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d(kernel_size=(2, 2), stride=2)

        self.conv3 = nn.Conv2d(64, 128, (4, 4), stride=2, padding=1)
        self.convnorm3 = nn.BatchNorm2d(128)
        self.pool3 = nn.MaxPool2d(kernel_size=(2, 2), stride=2)

        self.linear1 = nn.Linear(128*1*1, 1024)
        self.linear1_bn = nn.BatchNorm1d(1024)
        self.drop = nn.Dropout(DROPOUT)
        self.linear2 = nn.Linear(1024, 149)

        self.relu = torch.relu

    def forward(self, x):
        x = self.pool1(self.convnorm1(self.relu(self.conv1(x))))
        x = self.pool2(self.convnorm2(self.relu(self.conv2(x))))
        x = self.pool3(self.convnorm3(self.relu(self.conv3(x))))
        x = self.drop(self.linear1_bn(self.relu(self.linear1(x.view(len(x), -1)))))
        x = self.linear2(x)
        return x
```

Figure 1b. The data loader and simple CNN Mary 2 code.

Afterwards, I decided to adopt the pre-trained models DenseNet-161 and MobileNetV2. For DenseNet-161, I froze the layers within the first child attribute to take advantage of the pretrained parameters. I ran the frozen layers sequentially followed by a single linear layer that outputs a 1-D array of size 149, which is the total number of unique labels, which can be found in densenet_model.py (Figure 2).

```
def create_data_loader(img_dir, info_csv_path, batch_size):
    """Returns a data loader for the model."""
    img_transform = transforms.Compose([transforms.Resize(256),
                                        transforms.CenterCrop(224),
                                        transforms.ToTensor()])
    img_dataset = FashionDataset(img_dir, img_transform, info_csv_path)
    data_loader = DataLoader(img_dataset, batch_size=batch_size, shuffle=False, num_workers=12, pin_memory=True)
    return data_loader


MODEL_NAME = "densenet_model"


class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.densenet_model = models.densenet161(pretrained=True)
        for child in self.densenet_model.children():
            for param in child.parameters():
                param.requires_grad = False
        self.features = nn.Sequential(*list(self.densenet_model.children())[:-1])
        self.linear = nn.Linear(108192, 149)

    def forward(self, x):
        x = self.features(x)
        x = self.linear(x.view(len(x), -1))
        return x
```

Figure 2. The data loader and DenseNet-161 code.

Due to time constraints, I decided to not train DenseNet-161 on the fashion dataset and test it on our test set. For MobileNetV2, I froze the layers within the first child attribute to take advantage of the pretrained parameters. I ran the frozen layers sequentially followed by a single linear layer, trained on our fashion dataset, that outputs a 1-D array of size 149, which is the total number of unique labels, which can be found in mobilenet_model.py (Figure 3).

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.mobile_model = models.mobilenet_v2(pretrained=True)
        n = 0
        for child in self.mobile_model.children():
            n += 1
            if n < 2:
                for param in child.parameters():
                    param.requires_grad = False
        self.features = nn.Sequential(*list(self.mobile_model.children())[:-1])
        self.linear = nn.Linear(62720, 149)

    def forward(self, x):
        x = self.features(x)
        x = self.linear(x.view(len(x), -1))
        return x
```

Figure 3. The data loader and MobileNetV2 code.

Regarding fashion recommendations, I decided to try different KNN methods, which can be found in baseline_recommendations.py and generate_recommendations.py. First, I tried the scikit-learn BallTree method with Euclidean distance, which appeared to perform well (sklearn.neighbors.BallTree, n.d.; Tuinhof, Pirker, & Haltmeier, 2018). Second, I attempted to use annoy for approximate nearest neighbors with cosine similarity. However, this method did not work well. For different example images, this method would return the exact same store image recommendations (Spotify, 2019). Therefore, I decided to utilize the scikit-learn BallTree method with Euclidean distance (Figure 4a-b).

```
# Scikit-learn KNN
with open("{}_{}_recommendations.txt".format(MODEL_NAME, EXAMPLE_TYPE), "w") as file:
    file.write("Model: {}, Example Type: {}, Scikit-learn KNN \n".format(MODEL_NAME, EXAMPLE_TYPE))
rng = np.random.RandomState(42)
tree = BallTree(store_feature_maps)
dist, ind = tree.query(example_feature_maps, k=5)
print(dist) # Distances to 5 closest neighbors
with open("{}_{}_recommendations.txt".format(MODEL_NAME, EXAMPLE_TYPE), "a") as file:
    file.write("Distance to 5 closest neighbors: {} \n".format(dist))
for i in ind:
    for idx in i:
        print(store_df['image_label'][idx])
        with open("{}_{}_recommendations.txt".format(MODEL_NAME, EXAMPLE_TYPE), "a") as file:
            file.write("Recommendation: {} \n".format(store_df['image_label'][idx]))
```

Figure 4a. The scikit-learn Ball-Tree KNN using Euclidean distance code.

```
# Annoy KNN
with open("{}_{}_recommendations.txt".format(MODEL_NAME, EXAMPLE_TYPE), "a") as file:
    file.write("Model: {}, Example Type: {}, Annoy KNN \n".format(MODEL_NAME, EXAMPLE_TYPE))
# Index store
store_item = AnnoyIndex(store_feature_maps.size()[1], 'angular')
for i in range(store_feature_maps.size()[0]):
    store_item.add_item(i, store_feature_maps[i])
store_item.build(500) # More trees gives higher precision when querying
store_item.save('store_items.ann')
# Index example
example_item = AnnoyIndex(example_feature_maps.size()[1], 'angular')
example_item.load('store_items.ann')
recommendations = example_item.get_nns_by_item(0, 5)
dist_recommendations = example_item.get_nns_by_item(0, 5, include_distances=True)
print(dist_recommendations)
with open("{}_{}_recommendations.txt".format(MODEL_NAME, EXAMPLE_TYPE), "a") as file:
    file.write("Distance to 5 closest neighbors: {} \n".format(dist_recommendations))
for recommendation in recommendations:
    print(store_df['image_label'][recommendation])
    with open("{}_{}_recommendations.txt".format(MODEL_NAME, EXAMPLE_TYPE), "a") as file:
        file.write("Recommendation: {} \n".format(store_df['image_label'][recommendation]))
```

Figure 4b. The annoy KNN using cosine similarity code.

Results

The two simple CNN models that I created, Mary 1 and Mary 2, did not outperform the simple CNN model Jessica 5 on the test set, obtaining an average micro-averaged F1-score of 0.27 and 0.29, respectively. Concerning DenseNet-161, the micro-averaged average F1 score was 0.060, the maximum was 0.070 and the minimum was 0.050 (Figure 5).

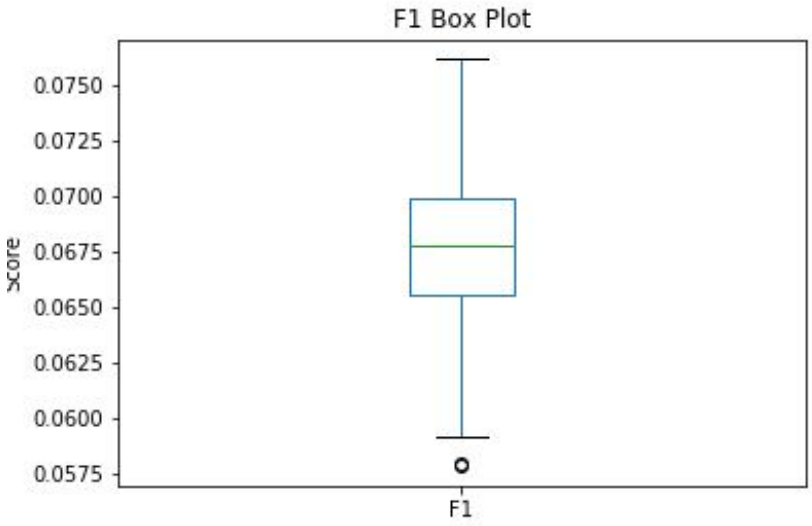

Figure 5. Micro-averaged F1-score distribution for DenseNet-161.

As expected, not training DenseNet-161 on the fashion dataset produced subpar results. Regarding MobileNetV2, the training loss indicates that the loss significantly decreased in the beginning of training and flattened out for the rest of training (Figure 6).

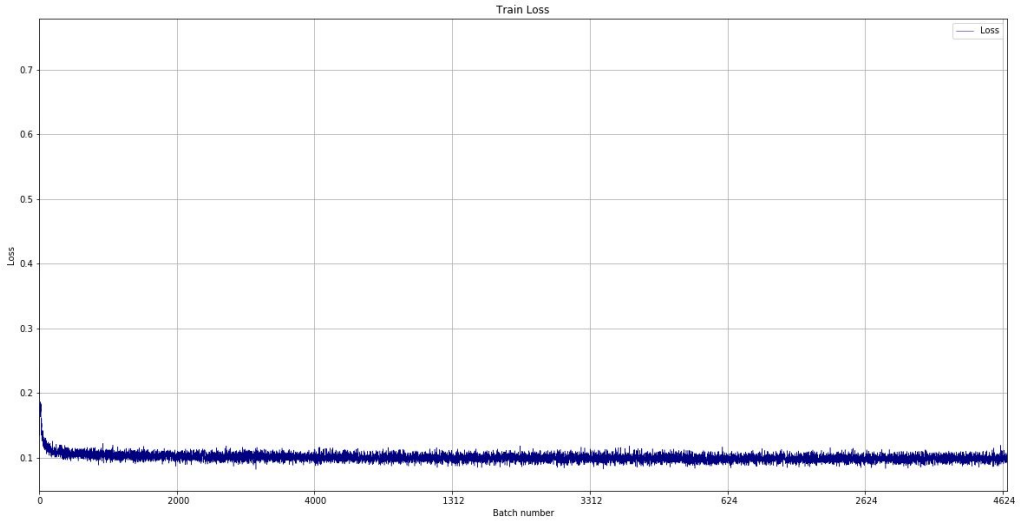

Figure 6. The training loss for MobileNetV2.

This suggests that MobileNetV2 may possibly be overfit, but for our use case of fashion recommendation, overfitting may not be a major issue. MobileNetV2 performed slightly better than the simple CNN model Jessica 5 in terms of multilabel classification. The average F1 score was 0.396, the maximum was 0.417 and the minimum was 0.368. (Figure 7).

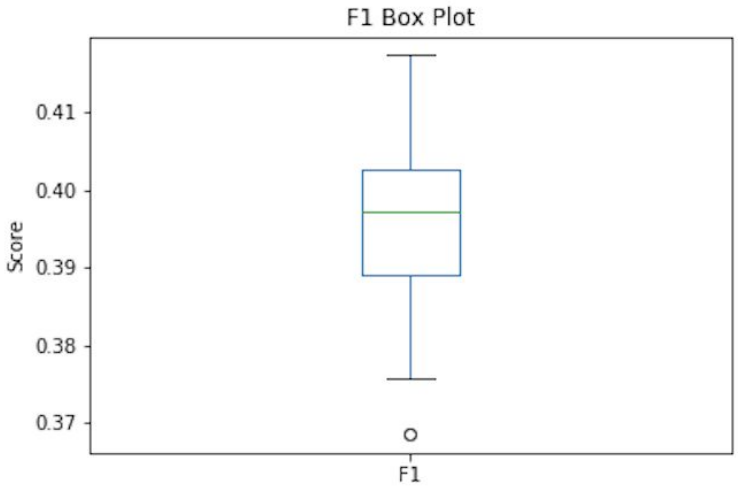

Figure 7. Micro-averaged F1-score distribution for MobileNetV2.

Below, are examples of recommendations using the scikit-learn BallTree method with Euclidean distance. The first example query image that I tried was a pair of jeans (Figure 8).

Figure 8. Jeans recommendations every model.

From this example, it is apparent that the baseline KNN recommendations do not look similar to my example query image. However, the simple CNN model Jessica 5 and MobileNetV2 feature extraction followed by KNN ranked recommendations appeared to perform well, recommending store images containing jeans. MobileNetV2 performed better than the simple CNN because it picked up on the faded, light pattern near the knees. The second example image that I tried was a skirt (Figure 9).

Figure 9. Skirt recommendations for every model.

From this example, it is evident that all of the models picked up on the color of the pink skirt and the nude shoes. However, the baseline KNN and the simple CNN model Jessica 5 did not pick up on the cut and the length of the skirt well, and provided recommendations of pants. The MobileNetV2 recommendations are superior to the others since they not only picked up on the colors within the example query image, but also the cut and length of the skirt well. For the last example, I decided to observe how well our fashion recommendation models would transfer to real-life images. Therefore, I decided to use an example query image consisting of Jessica and her dog, Conrad (Figure 10).

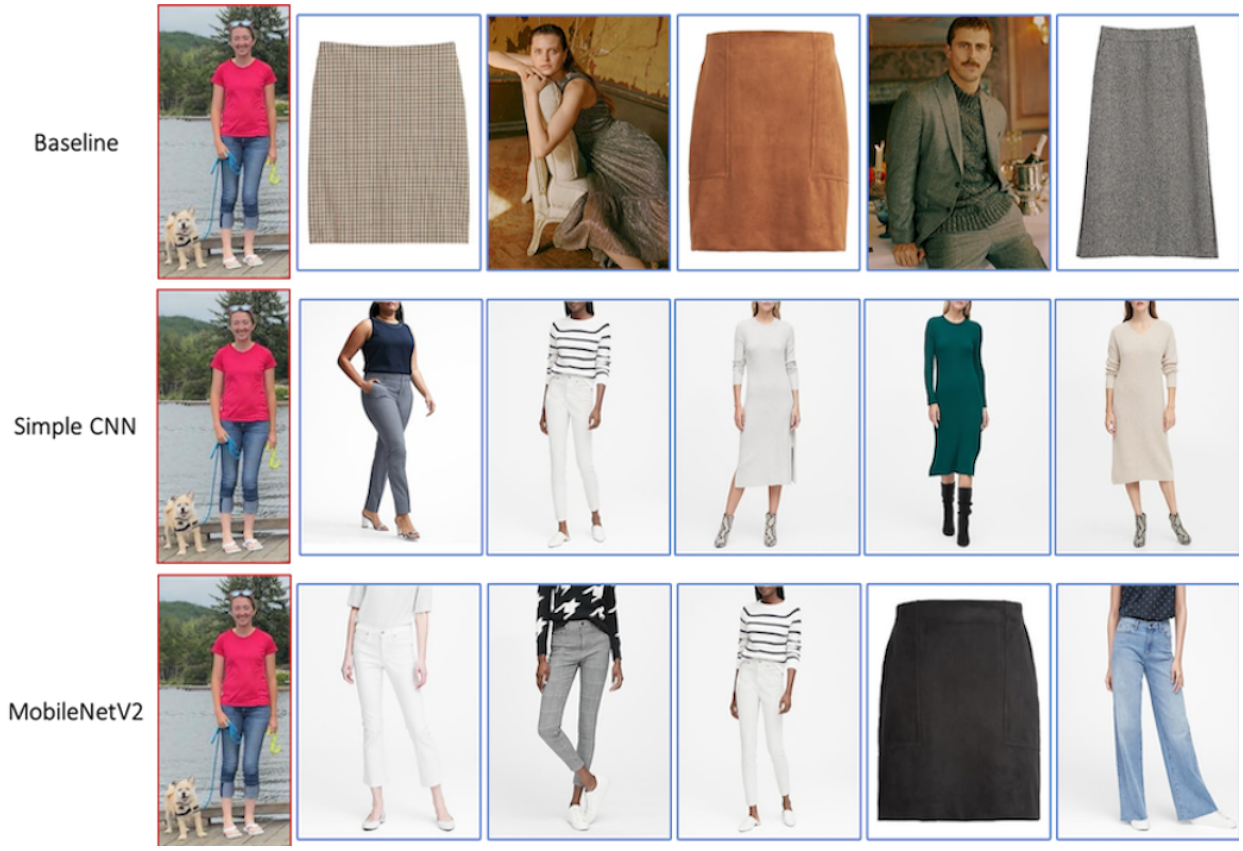

Figure 10. Jessica recommendations for every model.

It is apparent that the models did not perform extremely well on the Jessica example image. Based on the recommended images, it is evident that the models picked up on the colors within the example image, such as the green trees, bluish-grey water, and black harness on Conrad. I want to point out that the simple CNN model Jessica 5 and MobileNetV2 made a lot of recommendations related to white clothing, picking up on the white shoes within the example query image. Based on the word cloud, white is one of the majority labels in the training set, indicating that the label imbalance within the training set might be a cause for concern and something that I should address. Also, of note, the recommended images consisted of skirts, dresses, and pants, but not tops, which suggests the need for bounding boxes and pose estimation. Overall, I feel that MobileNetV2 performed the best for this task since it recommended pants that appear to be a similar length to the example query image.

Summary & Conclusions

Based on the proposed two-step approach, I first trained CNN models on fashion images in order to extract the feature maps associated with different clothing items. Second, I used those feature maps to create a KNN model that returns the five nearest neighbors to a given query image as clothing recommendations. I found that the CNN model MobileNetV2 produced the best results

in terms of micro-averaged F1-score and when coupled with the the scikit-learn Ball-Tree KNN model using Euclidean distance produced the best fashion recommendations. From this project, I learned that having an amazing partner and organization are key to successfully completing group projects. Also, I learned that CNN models can be used for feature extraction and that KNN can be utilized for query-based ranking. Moreover, fashion recommendation is a difficult problem and our solution is far from optimal. To improve upon this project, I would deal with the label imbalance and experiment with the DeepFashion and DeepFashion2 datasets. Both of these datasets include bounding boxes, which I would like to use to extract feature maps for individual clothing items, and pose estimations, which will aid in dealing with different poses within fashion images that may help to better generalize our approach to real-life images. Moreover, I would like to assess future models by utilizing Top-k accuracy since I am only interested in the first k items being correct and relevant in order to produce a recommendation.

Code Calculation

Calculating the percentage of open-source code from the internet that you utilize in a project is ridiculous. I'm sure that when you are working on projects that you don't keep track of every single line that you find or copy from the internet. According to my calculations, 40% of the code found within my contributions to this project I found or copied from the internet. I cannot confirm nor deny that this is correct.

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.