Final Project Report: Machine Learning II
Submitted to: A. Jafari
By Group 9
Jason Achonu, Kyi Win, Manoah Farmer

Project Title
Image Classification on Online Stores Fashion Dataset

1. Introduction

One of the interesting areas that Artificial Intelligences specifically computer vision has been applied to recently is fashion and searching clothing items.

The aim of this project is to build a Convolution Neural Network to Identify different types of online clothing items. The model will be evaluated with the following metrics: accuracy, F1 score, precision and recall, as well as AUC and ROC curves. To achieve this we will be tuning hyperparameters such as the number of epochs, number of layers, kernel size, etc. to improve our model.

This report's contents are as follows:Description of our data, Description of algorithm, details of experimentation methods, Results and Summary of what we have learned.

2. Description of the dataset.

Our Data was originally downloaded by University of North Carolina at Chapel Hill students. It consisted of 404,683 images of clothing from different online stores: Threalreal, Amazon, 6pm, Modcloth, Neimanmarcus, Nordstrom, Asos, Zappos, Shopbop, Macy's, Bloomingdale's, Forever 21, DSW, Target, Kohl's, JC Penny's, Urban Outfitters, Anthropologie, J.Crew, Banana Republic, Old Navy, Express, Piper Lime, Gap, and Factory J.Crew. Within this dataset, there are eleven categories that the images fall under: bags, belts, dresses, eyewear, footwear, hats, leggings, outerwear, pants, skirts, and tops. Because this data would take a extremely large amount of time to train and run, we decided to limit the amount of data that we took into account. We did this by writing a code to choose a reduced number of images from six categories that contained the largest number of images. The six resulting categories are as follows: tops, skirts, bags, outerwear, dresses, and footwear. We ended up with a total of 72,369 images.

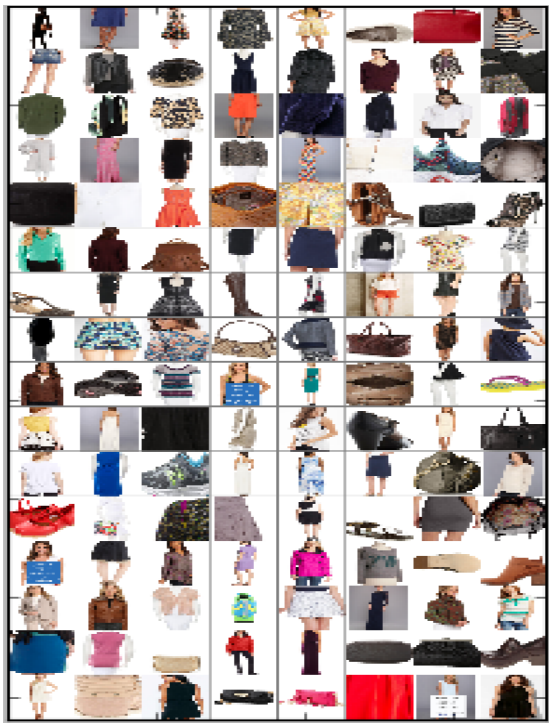


Figure 1: Sample of clothing items

Data Preprocessing

After downloading, the images were split into train and test sets, 70% in training and 30% in test. The Images were then converted to Tensors using Pytorch, and were resized to 256 pixels. We then used centercrop, which left us with 224x224 pixel images. Next, we normalized our dataset using the uniform distribution of 0.5.

3. Description of algorithm.

Provide some background information on the development of the algorithm and include necessary equations and figures.

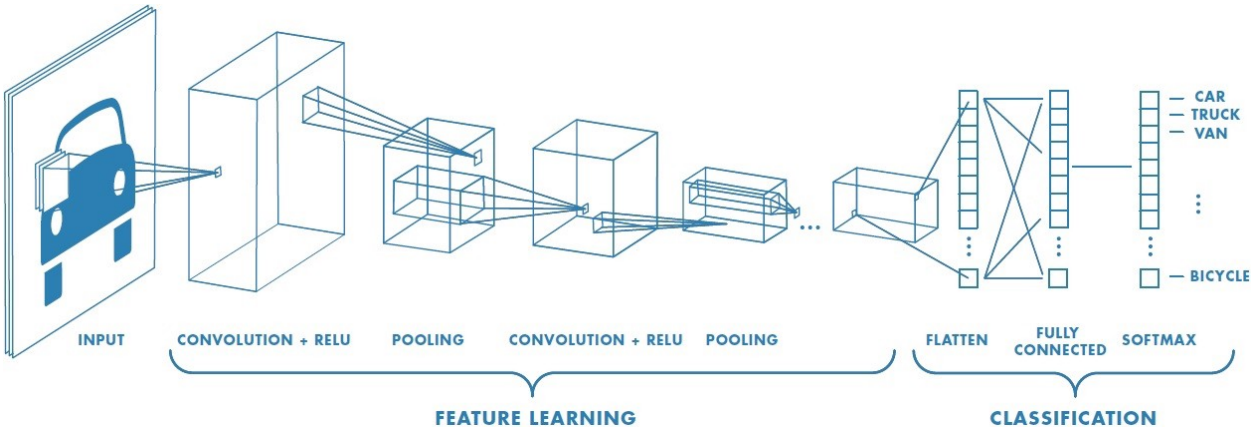


Figure 2 : Convolution Neural Network with Fully connected layers

Convolution Layer

Convolutional neural network:
This network is a multilayer feedforward neural network that has two, or three dimensional inputs. The primary layer for this network is the convolutional layer and and the weight functions of this network perform convolution operation on the image, using the convolution kernel. This is done while still preserving the relationship between pixels. To insure that our code runs correctly the following equation was used to determine feature-map size:

$$z_{i,j} = \sum_{k=1}^{r} \sum_{l=1}^{c} w_{k,l} v_{i+k-1,j+l-1}$$

The bottleneck approach in deep learning is used to reduce the dimensionality of our images to account for noise amongst the dataset. This approach has been disputed by some data scientists,but remains a reliable tool for others. We decided to explore its use and determine for ourselves whether the bottleneck approach is suitable for this dataset.

4. Experimental setup.

We chose to implement a CNN model using the pytorch framework. We started with only two convolution layers, and gradually increased to 3, 5 and seven respectively. We chose cross-entropy loss as our performance index. For all cases we used mini-batching with a batch-size of 64. We decided on this size because a size of 32 resulted in a long runtime and 64 is a commonly used batch-size in convolution. Had we used a larger batch-size computational time would remain high due to the model having to process a large amount of data at one time. We arbitrarily decided on the value of 0.01 for our learning rate initially, and later tested our model

with a learning rate of 0.1, which resulted in our accuracy reducing drastically. For the majority of the time we spent on our project, we did not experience overfitting in our model. In this project, we also implemented the bottleneck feature in an attempt to improve our model by reducing the feature-map size for a layer and then raising it again to compress the images and reduce noise while maintaining spatial features of the images.

5. Results.

With Channel of 3 and the same random sample overtime with set.seed (1122), our results are summarized as below.

Epoch	Iteration	Batchsize	Layers	Accuracy	Time	Image Size	Optimizer
1	4	100	2 (straight) 2 (maxpool)	65%	--	28x28	Adam
1	4	100	3 (straight) 2 (maxpool)	56%	104.281 539 sec	224x224	Adam
2	7	64	5 (bottleneck) 4 (maxpool)	65%	11121.9 87646 sec	224x224	Adam
4	7	64	5 (bottleneck) 4 (maxpool)	70%	422.839 98 sec	224x224	Adam
5	7	64	5 (bottleneck) 4 (maxpool)	72%	717.425 04 sec	224x224	Adam
5	7	64	5 (straight) 4 (maxpool)	76%	795.74 sec	224x224	Adam
5	7	64	5 (bottleneck) 4 (maxpool)	80%	473.503 sec	224x224	SGD
5	7	64	5 (bottleneck) 4 (maxpool)	79%	481.23 sec	224x224	Adagrad
10	7	64	5 (bottleneck) 4 (maxpool)	79%	1542.55 6 sec	224x224	SGD
10	7	64	5 (bottleneck) 4 (maxpool)	81%	972.23 sec	224x224	Adagrad

5	7	64	5 (bottleneck) 4 (maxpool)	54%	1025.02 9 sec	224x224	Rprop
10	7	64	7 (straight) 4 (maxpool)	71%	1727.82 sec	224x224	Adam
10	7	64	7(bottleneck) 4(maxpool)	69%	1511.42 6 sec	224x224	Adam

With channel of 3 and set.seed to 1122, we tuned different hyperparameters to see changes that can cause to the results.

From the results we can interpret that the accuracy rate goes up when the layers are "straight" instead of bottleneck. Also when comparing the time it takes, we finds that the time it takes increases when the layers are straight. There seems to be about 4% differences between straight and bottleneck approach. Therefore, for this dataset, the straight approach performs better than the bottleneck approach and we find that straight approach isn't preferable in this image classification. The models that have the "straight" specification were run with identical layers with the exception of the max-pooling function which were only applied to the the four outermost layers to avoid feature map size issues.

Changing kernel-size resulted in a lower accuracy of the model for our 7 layer model. The models that have the "straight" specification were run with identical layers with the exception of the max-pooling function which were only applied to the the four outermost layers to avoid feature map size issues. Changing the optimizer to SGD changes the result of the performance and increase the performance of our model. The model accuracy rate increases from 72% to 80%, which is a significant 8% increase.

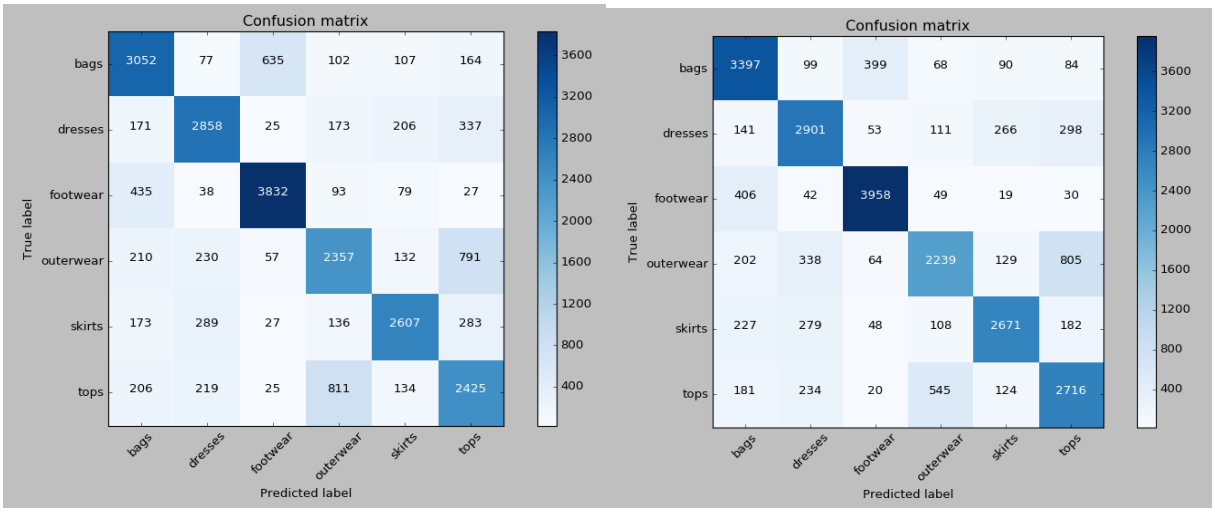
The total time to run the model decreases with the change in the optimizer as well. The total time with the Adam optimizer was 795.74 sec and the total time with the SGD optimizer was 473.50 while keeping all the other parameters the same, which shows that SGD optimizer not only increases the performance of the model accuracy but also saves the time it took to run the model.

SGD, Stochastic gradient descent optimizer is called stochastic because samples are selected randomly instead of a single group. While it performs better than Adam, increasing the layer using the SGD doesn't improve the model performance, and the total time took longer.

Confusion Matrix

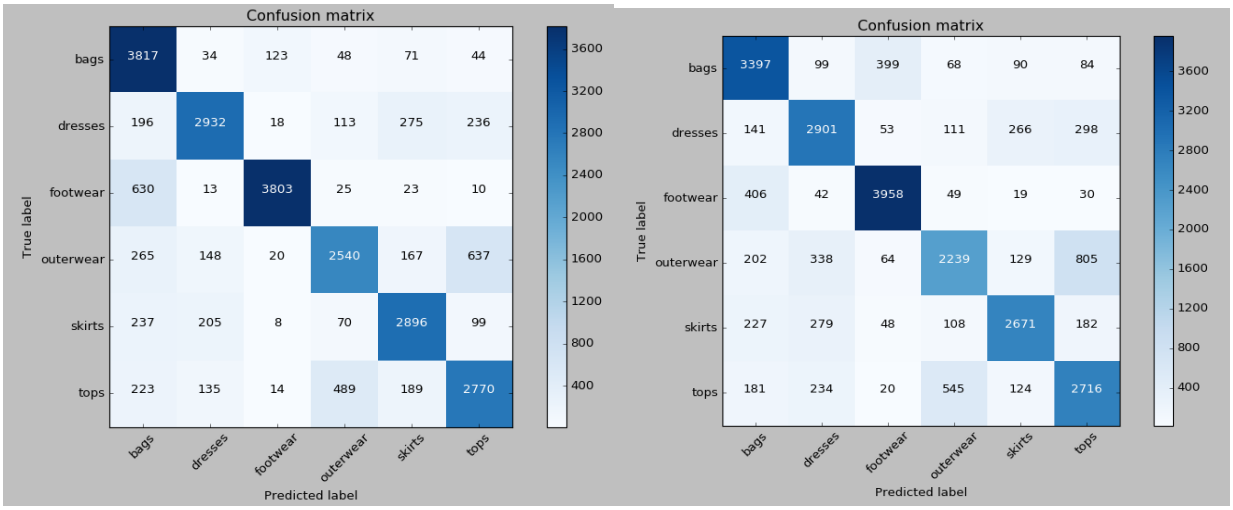
The bottom two graphs are confusion matrix obtained comparing the bottleneck and straight approach with 5 layers convolutional neural network with Adam optimizer. From the diagonal line, we can observed that the straight approach with 5 layers has better accuracy.

(5 layers left-bottleneck, right-straight)



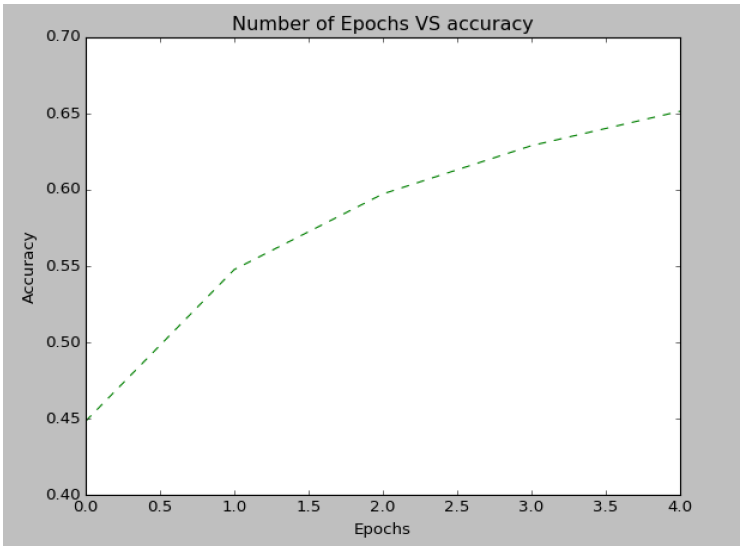
When comparing the confusion matrix models using of SGD optimizer and Adam optimizer with straight technique, we find that SGD optimizer has accuracy scores that are higher than the accuracy scores of Adam optimizer.

(straight SGD 5 layers, and straight Adam 7 layers)



Accuray vs Epoch

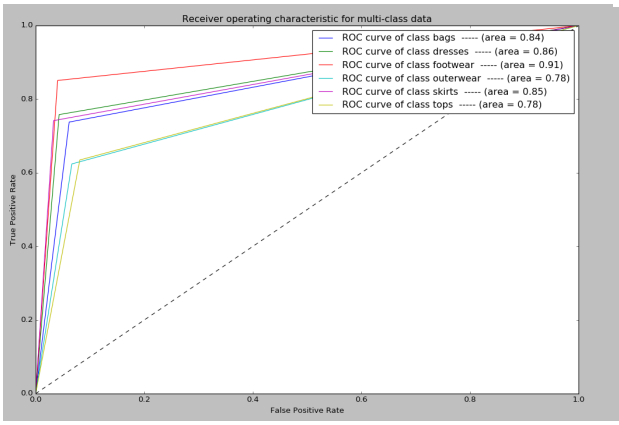
From the bottleneck approach with 5 layers, we also observed that the accuracy rates increase with the increase in the number of epoch from 0.45 to 0.65 in this case.

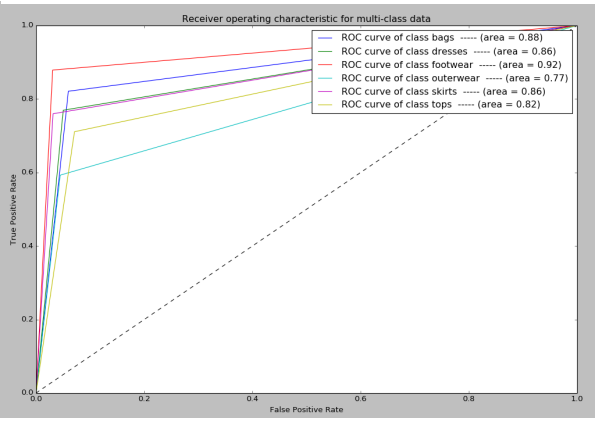


The ROC curve

Receiver Operating Characteristic (ROC) curve is defined as "the true positive rate (Sensitivity) is plotted function of the false positive rate (100-Specificity) for different cut-off points". From the two graphs below, we can see that the right graph has the curve which is closer to the upper corner and we can interpret that it has more accuracy. On the other hand, we can also interpret using the area under the curve (AUC) to determine and compare the accuracy of this bottleneck and straight approach. The areas under the graphs are specified in the box located at the upper corner for each item, and we find that the footwears has the highest accuracy in our image classification when comparing the 5 layers with bottleneck technique and straight technique.

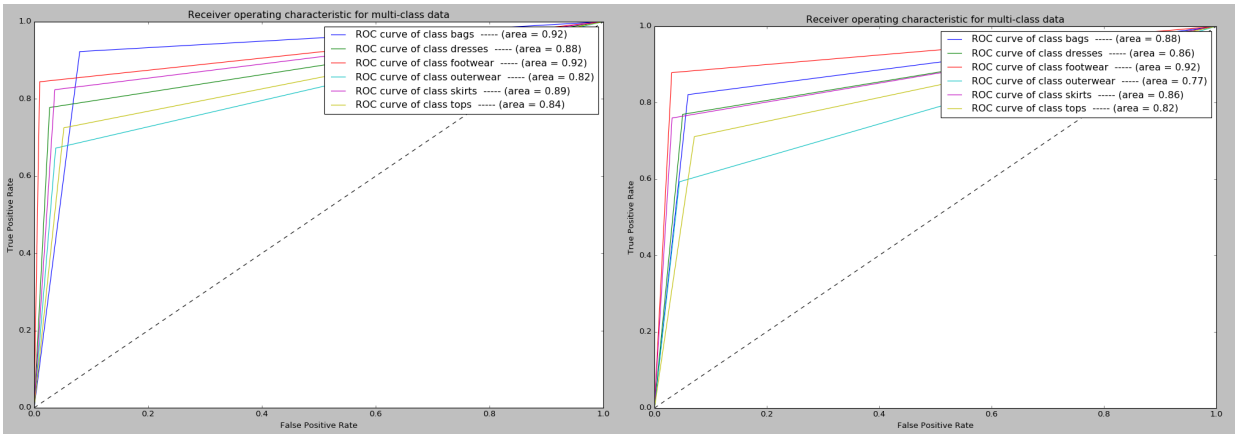
(left-bottleneck, right-straight)





The two graphs below shows the SGD and Adam ROC graphs and the graphs show that SGD performs better than Adam optimizer.

(left-SGD, right-Adam)



Accuray rates on each class (SGD best model)

Test Accuracy of the model on the test images: 80 %

Accuracy of bags : 85 %
Accuracy of dresses : 83 %
Accuracy of footwear : 90 %
Accuracy of outerwear : 74 %
Accuracy of skirts : 88 %
Accuracy of tops : 62 %

Classification Report (SGD best model)

	precision	recall	f1-score	support
bags	0.84	0.85	0.85	4137
dresses	0.81	0.79	0.80	3770
footwear	0.94	0.89	0.92	4504
outerwear	0.69	0.77	0.73	3777
skirts	0.75	0.86	0.80	3515
tops	0.79	0.65	0.71	3820
avg / total	0.81	0.81	0.81	23523

{0: 0.9078810302987728, 1: 0.8789473337898108, 2: 0.9406002873447017, 3: 0.8529724442436417, 4: 0.9042918181233909, 5: 0.8077317271428355}

From the classification report of the model with SGD optimizer, we can observed that the f1 score of the footwear is the highest among all the classes. Precision represents how precise or

accurate our model is - it says out of those predicted positive, how many of them are actual positive. In footwear, the precision is 94%. Recall calculates how many of the Actual Positive our model capture through labeling it as Positive. We can interpret from this results that this model classified 94% correctly out of all the footwear images that are labeled as footwear and out of all the footwear images, it classified 89% of images with accurate labels.

6. Summary and conclusions.

When observing the results of our experiments, we found that footwear, bags and skirts consistently performed the best in all of the experiments we conducted out of the categories we chose. In conclusion, the models that we ran performed better than random guess; however, there are ways this model could be improved going forward. Our best performing model was the one using the SGD optimizer. One way to do this would be to simply gather more data and possessing more processing power. We could also try tuning the hyper-parameter differently.

7. References.

https://forums.fast.ai/t/image-normalization-in-pytorch/7534
https://stackoverflow.com/questions/47850280/fastest-way-to-compute-image-dataset-channel-wise-mean-and-standard-deviation-in
http://acberg.com/papers/wheretobuyit2015iccv.pdf
https://github.com/flipkart-incubator/fk-visual-search
https://github.com/Airconaaron/blog_post_visualizing_pytorch_cnn/blob/master/Visualizing%20Learned%20Filters%20in%20PyTorch.ipynb
https://arxiv.org/pdf/1311.2901.pdf
https://stackoverflow.com/questions/47850280/fastest-way-to-compute-image-dataset-channel-wise-mean-and-standard-deviation-in
https://github.com/amir-jafari/Deep-Learning/blob/master/Pytorch_/6-Conv_Mnist/Conv_Mnist.py
https://discuss.pytorch.org/t/data-augmentation-in-pytorch/7925
https://stackoverflow.com/questions/2301789/read-a-file-in-reverse-order-using-python
https://pytorch.org/docs/stable/torchvision/transforms.html

https://stackoverflow.com/questions/12984426/python-pil-ioerror-image-file-truncated-with-big-images
http://www.apsipa.org/proceedings/2017/CONTENTS/papers2017/14DecThursday/Poster%204/TP-P4.14.pdf
https://www.quora.com/How-can-I-calculate-the-size-of-output-of-convolutional-layer
https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
https://github.com/pumpikano/street2shop
https://stackoverflow.com/questions/123198/how-do-i-copy-a-file-in-python

8. Appendix.

Use “wget https://storage.googleapis.com/group8project2jkm/Dataset.zip” to download the data set and unzip the folder.
Download the data set.
The download.py script is used by first making the directory called “images” using the command “mkdir images” followed by running “python download.py --urls photos/photos.txt” outside of the images directory. This downloads the images into a folder called “images”.
Next run the code to split the data into categories by running “python split_image_train_test.py”. After splititng we ran the the cnn model code using “python Final_cuda.py”. Be sure to run this outside of the images and photos directory.

```
#coding: utf-8

from tomorrow import threads
import imghdr
import requests
import os
```

```
from itertools import islice
import sys
import logging


def split(x):
    first = x.find(',')
    return (x[:first], x[first+1:])


def main(args):

    if args.fail_file is not None:
        logging.basicConfig(filename=args.fail_file, format='%(message)s', level=logging.ERROR)



    @threads(args.threads)
    def download(item_id, url, i, images_dir=''):

        try:
            r = requests.get(url)

            if r.status_code == 200:

                image_type = imghdr.what(None, r.content)

                if image_type is not None:
                    with open(os.path.join(images_dir, item_id + '.' + image_type), 'wb') as f:
                        f.write(r.content)
                        f.close()
                else:
                    logging.error('%s\t%s\tunknown_type' % (item_id, url))
            else:
                logging.error('%s\t%s\tstatus:%d' % (item_id, url, r.status_code))

        except KeyboardException:
            raise
        except:
            print "Unexpected error:", sys.exc_info()[0]
            logging.error(sys.exc_info()[0])

        if i % 200 == 0:
            print i
```

```
    f = open(args.urls)

    itr = enumerate(f)
    itr = islice(itr, args.start, None)

    for i, line in itr:
        [item_id, url] = split(line.strip())
        download(item_id, url, i, images_dir=args.images_dir)



if __name__ == '__main__':
    from argparse import ArgumentParser
    parser = ArgumentParser()

    # Data handling parameters
    parser.add_argument('--urls', dest='urls', type=str, default=None, required=True, help='urls')
    parser.add_argument('--image_dir', dest='images_dir', type=str, default='images',
help='image directory')
    parser.add_argument('--failures', dest='fail_file', type=str, default=None, help='failure
records')
    parser.add_argument('--start', dest='start', type=int, default=0, help='start offset')
    parser.add_argument('--threads', dest='threads', type=int, default=10, help='threads')

    args = parser.parse_args()


    main(args)

    exit(0)


# coding: utf-8


import pandas as pd
import numpy as np
```

```
import shutil
import os
import os.path
from os import listdir
from os.path import isfile, join
import pandas as pd
from PIL import Image
import pathlib



cwd = os.getcwd()




meta_json = cwd + '/meta/json'



from os import listdir
from os.path import isfile, join
meta_json_files = [f for f in listdir(meta_json) if isfile(join(meta_json, f))]



listdir(meta_json)


meta_json_files.remove('.DS_Store')


image_dir = cwd + '/images'



image_files = [f for f in listdir(image_dir) if isfile(join(image_dir, f))]


len(image_files)
```

```
# Get all the images in each category



img_name_List = []
img_ext_List = []


for i in image_files:
    img_name = i.split('.',1)[0]
    img_ext = i.split('.',1)[1]

    img_name_List.append(img_name)

    img_ext_List.append(img_ext)



rr = pd.read_json(cwd+ '/meta/json/retrieval_outerwear.json')

photo_id = list(rr['photo'])



for file in meta_json_files:
    sub_folder_name_filename = file.split('_',1)[-1]
    #print(sub_folder_name_filename)
    sub_folder_name = sub_folder_name_filename.split('.')[0] #name of the folder
    #print(sub_folder_name)

    rr = pd.read_json(cwd+ '/meta/json/' +file)

    photo_id = list(rr['photo'])

    for idx in range(len(photo_id)):
        photo_id[idx] = str(photo_id[idx]).zfill(9)




    all_imgs = []
    #all_ext = []
```

```
for img_idx in photo_id:

    if img_idx in img_name_List:

        index = img_name_List.index(img_idx)



        all_imgs.append(img_idx + '.' + img_ext_List[index])

TRAIN_SET = all_imgs[len(all_imgs)//3:]

TEST_SET = all_imgs[:len(all_imgs)//3]




#  TRAIN SET
for img_data in TRAIN_SET:
    destination_folder = str(cwd) + '/Dataset/Train/' + str(sub_folder_name) # + str(img_data)


    if not os.path.exists(destination_folder):
        os.makedirs(destination_folder)


    q = pathlib.Path(destination_folder + '/' + str(img_data))

    shutil.copy(image_dir + '/' + img_data , q)




#  TEST SET
for img_data in TEST_SET:
    destination_folder = str(cwd) + '/Dataset/Test/' + str(sub_folder_name) # + str(img_data)


    if not os.path.exists(destination_folder):
        os.makedirs(destination_folder)


    q = pathlib.Path(destination_folder + '/' + str(img_data))
```

```
        shutil.copy(image_dir + '/' + img_data , q)



# coding: utf-8

import torchvision.transforms as tt
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
import time
import torch.nn as nn
import torch
import torchvision
import torch.nn as nn
from torch.autograd import Variable
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from PIL import ImageFile
import itertools

from torchvision import models

ImageFile.LOAD_TRUNCATED_IMAGES = True
torch.manual_seed(1122)
#-------------------------------------------------------------------------------
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
#-------------------------------------------------------------------------------
transforms = transforms.Compose([transforms.Resize((256,256)), transforms.CenterCrop(224),
transforms.ToTensor(),transforms.Normalize(mean=[0.5, 0.5, 0.5],std=[0.5, 0.5, 0.5])])


batch_size = 64


data_path = 'Dataset/Train/'
train_dataset = torchvision.datasets.ImageFolder(
root=data_path,
transform=transforms
```

```
)
train_loader = torch.utils.data.DataLoader(dataset = train_dataset,
batch_size= batch_size,
shuffle=True, num_workers=16

)


data_path = 'Dataset/Test/'
test_dataset = torchvision.datasets.ImageFolder(
root=data_path,
transform=transforms
)

test_loader = torch.utils.data.DataLoader( dataset = test_dataset,
batch_size= batch_size,
shuffle=False, num_workers=16
)

classes =('bags',  'dresses', 'footwear', 'outerwear', 'skirts', 'tops')


train_iter = iter(train_loader)
print(type(train_iter))

images, labels = train_iter.next()


print('images shape on batch size = {}'.format(images.size()))
print('labels shape on batch size = {}'.format(labels.size()))



import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
```

```
# get some random training images
dataiter = iter(train_loader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(batch_size)))
plt.show()

num_epochs = 10
learning_rate = 0.01

# CNN Model (2 conv layer)
class CNN(nn.Module):
    def __init__(self):

        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU())
        self.layer4 = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer5 = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.fc = nn.Linear(14 * 14 * 32, 1000)
```

```
        self.fc1 = nn.Linear(1000, 6)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.layer5(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        out = self.fc1(out)
        return out

cnn = CNN()
cnn.cuda()



# ----------------------------------------------------------------------------
# Loss and Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(cnn.parameters(), lr=learning_rate)
# ----------------------------------------------------------------------------
# Train the Model
start_time = time.clock()
all_train_accuracy = []
total = 0
correct = 0
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = Variable(images).to(device)
        labels = Variable(labels).to(device)

        # Forward + Backward + Optimize
        optimizer.zero_grad()
        outputs = cnn(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if (i + 1) % 100 == 0:
            print('Epoch [%d/%d], Iter [%d/%d] Loss: %.4f'
                  % (epoch + 1, num_epochs, i + 1, len(train_dataset) // batch_size, loss.item()))
```

```
    #Mine - Accuracy for training set
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted.cpu() == labels.cpu()).sum()

    all_train_accuracy.append(correct.cpu().numpy()/total)
# --------------------------------------------------------------------------------
print('Training set Accuracy of the model on the Train images: %d %%' % (100 * correct / total))
# --------------------------------------------------------------------------------
#Plot the accuracy for the num of epochs verus accuracy

plt.plot(range(num_epochs),all_train_accuracy, 'g--')
plt.title('Number of Epochs VS accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.show()

# --------------------------------------------------------------------------------
# Test the Model
cnn.eval()  # Change model to 'eval' mode (BN uses moving mean/var).
correct = 0
total = 0
for images, labels in test_loader:
    images = Variable(images).to(device)
    outputs = cnn(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels.to(device)).sum()
end_time = time.clock()
print(str(end_time-start_time) + " seconds")
# --------------------------------------------------------------------------------
print('Test Accuracy of the model on the test images: %d %%' % (100 * correct / total))
# --------------------------------------------------------------------------------
# Save the Trained Model
torch.save(cnn.state_dict(), 'cnn.pkl')
#


class_correct = list(0. for i in range(6))
class_total = list(0. for i in range(6))


true_label = []
```

```
predicted_label = []


for data in test_loader:
    images, labels = data
    images = Variable(images).to(device)
    outputs = cnn(images)
    _, predicted = torch.max(outputs.data, 1)

    # mine
    y_proba, predicted = torch.max(outputs.data, 1)
    # end mine

    c = (predicted.cpu() == labels)

        #Add true labels

    true_label.extend(labels.cpu().numpy())
    #Add predicted labels
    predicted_label.extend(predicted.cpu().numpy())



    for i in range(4):
        label = labels[i]
        class_correct[label] += c[i].item()
        class_total[label] += 1

# ----------------------------------------------------------------------------------------
for i in range(6):
    print('Accuracy of %5s : %2d %%' % (classes[i], 100 * class_correct[i] / class_total[i]))
# ----------------------------------------------------------------------------------------

#cnf_matrix = confusion_matrix(labels.cpu(), predicted.cpu())

#Change later
cnf_matrix = confusion_matrix(true_label,predicted_label)



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
```

```
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    print('Confusion matrix')

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

# Compute confusion matrix
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure(figsize=(10, 7))
plot_confusion_matrix(cnf_matrix, classes=classes,
                      title='Confusion matrix')

plt.show()


target_names = classes
print(classification_report(true_label,predicted_label, target_names=target_names))


#ROC and AUC


from sklearn.metrics import roc_curve, auc
```

```
#change the class to stop at 5
from sklearn.preprocessing import label_binarize
y_label = label_binarize(true_label, classes=[0, 1, 2, 3, 4, 5])



y_predict = label_binarize(predicted_label, classes=[0, 1, 2, 3, 4, 5])



n_classes = len(classes)




fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_label[:, i], y_predict[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
#colors = cycle(['blue', 'red', 'green'])



print(roc_auc)


plt.figure(figsize=(15,10))

for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label='ROC curve of class {0}  ----- (area = {1:0.2f})'
             ''.format(classes[i], roc_auc[i]))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic for multi-class data')
plt.legend()
plt.show()
```

```
# Test the model

size = 4
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=size,
                                          shuffle=False, num_workers=4
                                          )


dataiter = iter(test_loader)
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(size)))
plt.show()


images = Variable(images).to(device)
outputs = cnn(images)



_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(size)))
```