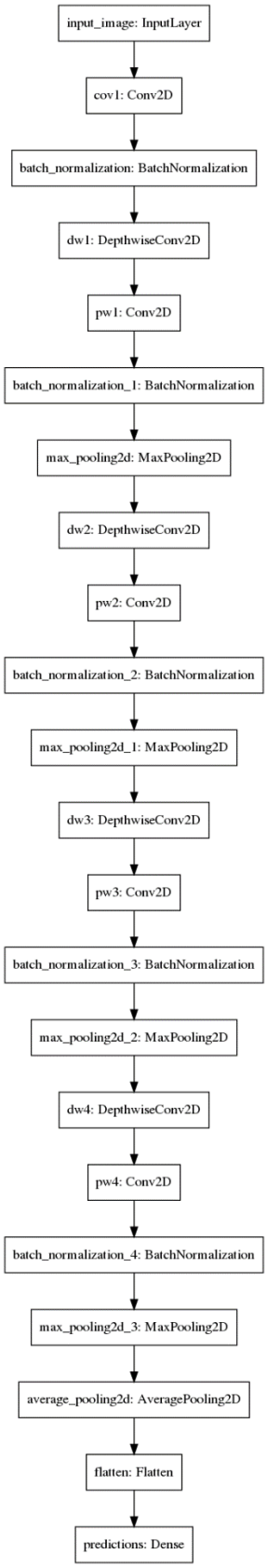
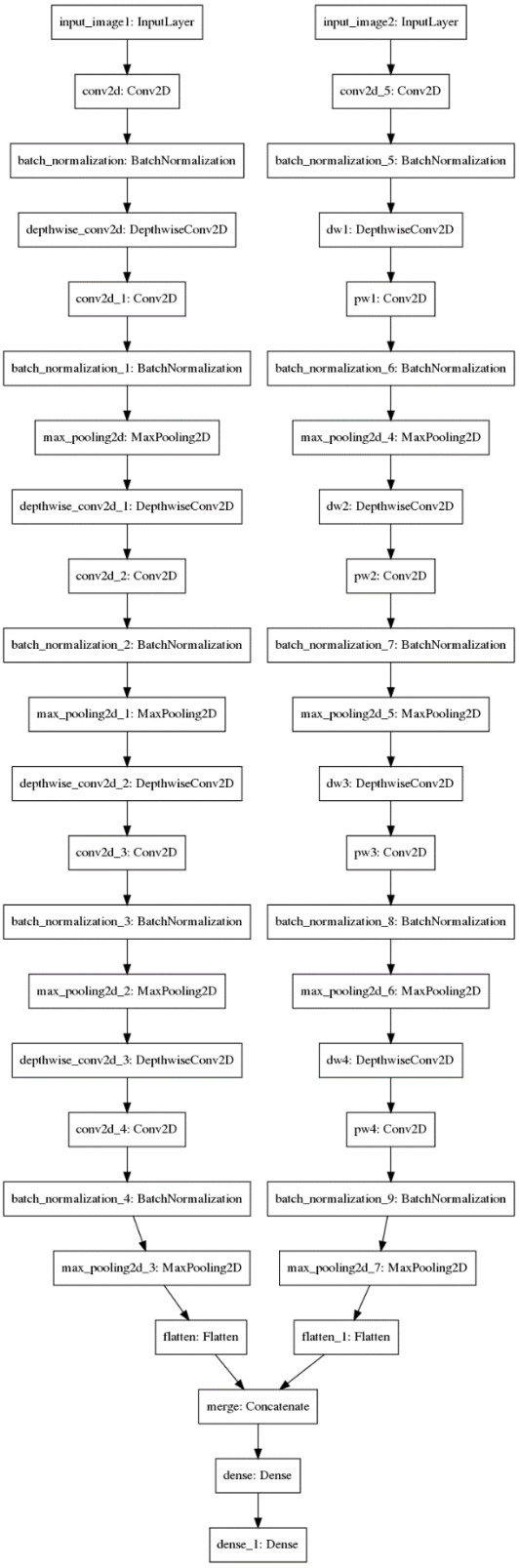
1. Model structure(the finetune model figure is too long and submitted separately)

The structure of own designed model:

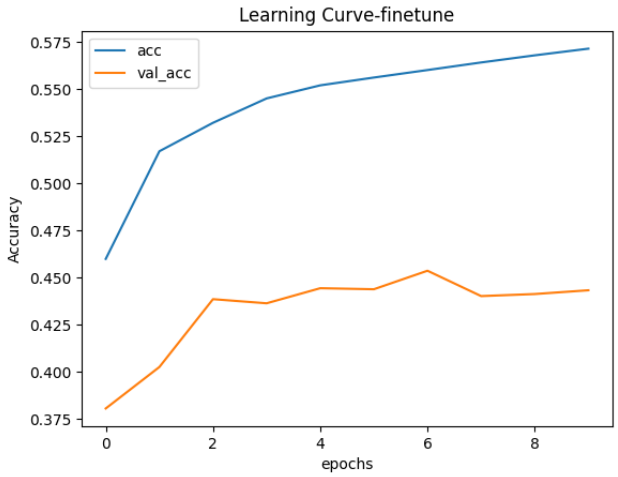



The structure of pairwise model

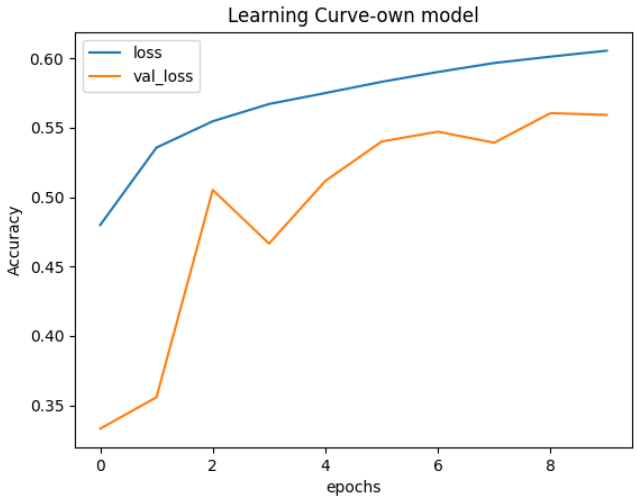


2.

Learning curve of category classification (totally 154 categories, use pretrained Mobile Net model):



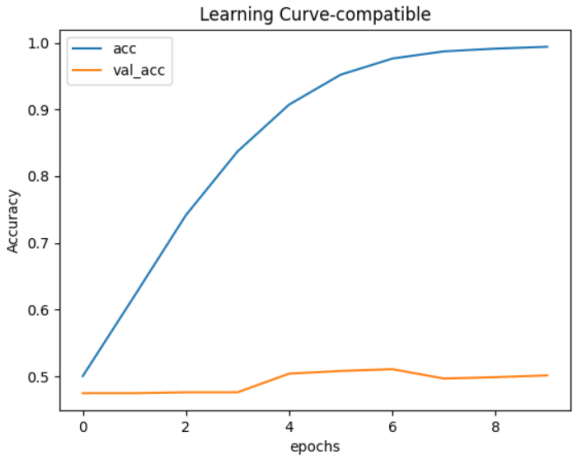

Learning curve of category classification, use own model:




When using pretrained model, the learning curve become smoother. Because the own-designed model train from initial weights, it needs more adjustment.

Also using pretrained model converges faster.

The learning curve of pairwise classification:




I have tried many groups of parameters (learning rate, batch size…) and most of them gets results of 0.5, for both training and validation dataset, very bad like it is choose randomly.
In the above curve, I only trained 10,000 data sets, randomly choose from around a million training data points, with 1200 validation data points. I use lr=0.00001 which is very small. Though the validation is still bad, the train accuracy curve seems pretty good. I thought it's because the training dataset is not enough. So, I increased the training data set to 100,0000 also increased the validation data set, and I ran three epochs. During the three epochs, I saw the training accuracy increased steadily, however, still I can't see steady increasement in validation accuracy. Because of the tight budget and limited time, I gave up to run to the end and plot the learning curve. And the above learning curve is the best I can get. I'll continue explore it if I got time and extra money after final project.

3.
The test results of category are in category_pred_hw.txt. The test results of pairwise compatible is in compatible_pred_hw.txt. I only choose first 80 rows among pair wise test data. Again, I hope to save some credits cause I have spent more than 70$ in this assignment.

4.
(1)
To test the fine tune model and own designed model for outfits category, firstly change the Config['root_path'] in utils.py corresponding to your polyvore_outfits folder path, secondly, change Config['test_file_path'] corresponding to the path of your test.txt. Then run test_category.py. The test results include both fine tune model and own designed model will in category_ownmodel_pred.txt and category_finetune_pred.txt in the same folder with test_category.py.
If you get error: can't find image directory. Probably because 16$^{th}$ line in test_category.py. When I read test_category_hw.txt, there is a '\n' character at the end of each row, so I use x[:-1] to exclude it. But when I read test_pairwise_compat_hw.txt, there is no '\n' at the end of row. If you get error can't find directory please kindly try to include all characters in the row by changing x[:-1] to x.

(2)
To test the pair wise compatibility, similarly change Config['root_path'] and Config['test_file_path'] in utils.py and run test_compatibility.py. The results will show in compatible_pred.txt in the same folder.
(3)
To test bonus assignment, change Config['root_path'] and Config['test_file_path'] in utils.py then run test_bonus.py. The results will show in bonus.txt in the same folder.

I think there is a bug in the assignment. There are two files in given polyvore_outfits folder. One is called test_pairwise_compat_hw.txt another called compatibility_test_hw. The former one gives test data based on pair-wise item id, the second one based on the whole set, including a group of items. So, the feed-in methods for test bonus and pair wise compatibility are different. I don't know I should used which one to test pair-wise compatibility, but I choose the first one, test_pairwise_compat_hw.txt. However, when test the compatibility of a group of figures, I can only use test_pairwise_compat_hw.txt. Otherwise there is no group information.

Code:
Code for own model

```
from tensorflow.keras.layers import AveragePooling2D, Dense, Conv2D, MaxPool2D, Flatten, Input,BatchNormalization, DepthwiseConv2D, Flatten
from tensorflow.keras.models import Model
from dataloader import polyvore_dataset
from utils import Config
import tensorflow.keras as tfk
import matplotlib.pyplot as plt
import numpy as np
import os
import os.path as osp
# from tensorflow.keras import regularizers

if __name__=='__main__':

    # data generators
    # dataset = polyvore_dataset()
    # trainList, valList, nClass = dataset.readMeta()
    # if Config['debug']:
    #     trainList = trainList[:6000]
    #     valList = valList[:700]
    # trainData = dataset.load(trainList, batchSize=Config['batch_size'])
    # valData = dataset.load(valList, batchSize=Config['batch_size'])
```

```
    # # reg_val = 0.01
    # # drop_rate = 0.15
    # # build model
    # x_input = Input(shape=(224, 224, 3), name='input_image')
    # x = Conv2D(32, (3,3),activation='relu', name='cov1')(x_input)
    # x = BatchNormalization()(x)
    # x = DepthwiseConv2D((3,3), name='dw1')(x)
    # x = Conv2D(64, (1,1),activation='relu', name='pw1')(x)
    # x = BatchNormalization()(x)
    # x = MaxPool2D()(x)

    # x = DepthwiseConv2D((3,3),name='dw2')(x)
    # x = Conv2D(128, (1,1),activation='relu', name='pw2')(x)
    # x = BatchNormalization()(x)
    # x = MaxPool2D()(x)

    # x = DepthwiseConv2D((3,3),name='dw3')(x)
    # x = Conv2D(256, (1,1),activation='relu', name='pw3')(x)
    # x = BatchNormalization()(x)
    # x = MaxPool2D()(x)

    # x = DepthwiseConv2D((3,3),name='dw4')(x)
    # x = Conv2D(512, (1,1),activation='relu', name='pw4')(x)
    # x = BatchNormalization()(x)
    # x = MaxPool2D()(x)

    # x = AveragePooling2D()(x)
    # x = Flatten()(x)

    # predictions = Dense(nClass, activation = 'softmax', name = 'predictions')(x)

    # model = Model(x_input, predictions)
    # optimizer = tfk.optimizers.RMSprop(learning_rate=Config['learning_rate'])
    # # define optimizers
    #     model.compile(optimizer=optimizer,     loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    # model.summary()

    # # # # training - num worker is obsolete now
    #     #     result     =     model.fit(trainData,     validation_data=valData,
epochs=Config['num_epochs'],shuffle=False)

    # # model.save('ownmodel.h5')
```

```
# # acc = result.history['accuracy']
# # val_acc = result.history['val_accuracy']
# # epochs = np.arange(len(acc))


# # plt.figure()
# # plt.plot(epochs, acc, label='loss')
# # plt.plot(epochs, val_acc, label='val_loss')
# # plt.xlabel('epochs')
# # plt.ylabel('Accuracy')
# # plt.title('Learning Curve-own model')
# # plt.legend()
# # plt.savefig("own_model.png")
# # plt.show()
# from tensorflow.keras.models import load_model
# model = load_model('ownmodel.h5')
# plot_model(model, to_file='model.png')

dataset = polyvore_dataset(train='False')
filepath = osp.join(Config['root_path'], 'test_category_hw.txt')
f = open(filepath, "r")
itemidlist=[]
for x in f:
    itemidlist.append(x[:-1])
# itemidlist = itemidlist[:2]
image_dir = osp.join(Config['root_path'], 'images')
testList = [osp.join(image_dir, x + '.jpg') for x in itemidlist]
testdata = dataset.load_test(testList)
from tensorflow.keras.models import load_model
model = load_model('ownmodel.h5')
temp = model.predict(testdata)
prediction_ownmodel = [np.argmax(i) for i in temp]
results = np.vstack((itemidlist,prediction_ownmodel))
results = results.T
f = open('ownmodel_pred.txt', 'w')
for row in results:
    temp = row[0]+' ' + row[1] + '\r\n'
    f.write(temp)

# model = load_model('fine_tunemodel.h5')
# temp = model.predict(testdata)
# prediction_finetune = [np.argmax(i) for i in temp]
# results = np.vstack((itemidlist,prediction_ownmodel))
```

```
    # results = results.T
    # f = open('finetune_pred.txt', 'w')
    # for row in results:
    #     temp = row[0]+' ' + row[1] + '\r\n'
    #     f.write(temp)
```

Utils:

```
import numpy as np
import os
import os.path as osp
import argparse

Config ={}
# you should replace it with your own root_path
#Config['root_path'] = 'F:\\599dl\\hw4\\polyvore_outfits'
Config['root_path'] = '/home/ubuntu/polyvore_outfits'
# Config['root_path'] = '/root/polyvore_outfits'
Config['meta_file'] = 'polyvore_item_metadata.json'
Config['checkpoint_path'] = ''


Config['use_cuda'] = True
Config['debug'] = False
Config['num_epochs'] = 10
Config['batch_size'] = 64

Config['learning_rate'] = 0.001
Config['num_workers'] = 2
```

Dataloader:

```
import tensorflow as tf
import os, json, random
import os.path as osp
from utils import Config

class polyvore_dataset:
    def __init__(self, train=True):
        self.root_dir = Config['root_path']
        self.image_dir = osp.join(self.root_dir, 'images')
        self.Train = train
    '''
    decode one image
    '''
```

```
    def decodeImg(self, path):
        image = tf.io.read_file(path)
        image = tf.image.decode_jpeg(image, channels=3)
        image = tf.image.convert_image_dtype(image, tf.float32)
        return image
    '''
    decode file name and return the raw image
    '''
    def process(self, pathAndLabel):

        x = tf.strings.split(pathAndLabel,';')
        image = self.decodeImg(x[0])

        image = tf.image.resize(image, (300, 300))
        image = tf.image.central_crop(image, 224 / 300)
        image = 2*image - 1    # value in [-1,1]

        return image, tf.strings.to_number(x[1])

    def process_test(self, pathAndLabel):
        image = self.decodeImg(pathAndLabel)
        image = tf.image.resize(image, (300, 300))
        image = tf.image.central_crop(image, 224 / 300)
        image = 2*image - 1    # value in [-1,1]

        return image
    '''
    load in data in a streaming fashion
    '''
    def load(self, fileList, batchSize=32):
        data = tf.data.Dataset.from_tensor_slices(fileList)
        data = data.map(self.process, num_parallel_calls=tf.data.experimental.AUTOTUNE)
        # data = data.cache()    #
        data = data.batch(batchSize)
        data = data.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
        return data

    def load_test(self, fileList):
        data = tf.data.Dataset.from_tensor_slices(fileList)
        data                                =                                data.map(self.process_test,
num_parallel_calls=tf.data.experimental.AUTOTUNE)
        data = data.batch(1)
        return data
```

```
    '''
    read in meta info
    '''
    def readMeta(self):
        dictionary = {'max': 0}
        def translate(idOld):
            if idOld not in dictionary:
                dictionary[idOld] = dictionary['max']
                dictionary['max'] = dictionary['max'] + 1
            return str(dictionary[idOld])
        meta = open(os.path.join(self.root_dir, 'polyvore_item_metadata.json'), 'r')
        meta = json.load(meta)
        nameAndId = [os.path.join(self.image_dir, name + '.jpg' + ';' +
translate(label['category_id'])) for name, label in
                     meta.items()]
        random.shuffle(nameAndId)
        idx = 1 - int(len(nameAndId) * 0.2) # 80/20 split
        return nameAndId[:idx], nameAndId[idx:], dictionary['max']+1
```

For Finetune Model, the utils.py and dataloader.py is the same with the own designed model, Only the train.py is different.

```
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.models import Model
from dataloader import polyvore_dataset
from utils import Config
from tensorflow.keras.applications import MobileNet
import tensorflow.keras as tfk
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import regularizers
import errno
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import load_model

if __name__=='__main__':

    # data generators
    dataset = polyvore_dataset()
    trainList, valList, nClass = dataset.readMeta()
    if Config['debug']:
        trainList = trainList[:50000]
        valList = valList[:5500]
    trainData = dataset.load(trainList, batchSize=Config['batch_size'])
    valData = dataset.load(valList, batchSize=Config['batch_size'])
```

```
reg_val = 0.01
# drop_rate = 0.1

# # build model
base_model = MobileNet(weights='imagenet', include_top=False)
x = base_model.output
x = GlobalAveragePooling2D()(x)
# x = Dropout(drop_rate)(x)
x = Dense(512, activation='relu', kernel_regularizer=regularizers.l2(l=reg_val))(x)
# x = Dense(512, activation='relu')(x)
# x = Dropout(drop_rate)(x)
predictions = Dense(nClass, activation = 'softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
    layer.trainable = False

optimizer = tfk.optimizers.RMSprop(Config['learning_rate'])
# define optimizers
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.summary()

#     # training - num worker is obsolete now
result = model.fit(trainData, validation_data=valData,
epochs=Config['num_epochs'],shuffle=False)

model.save('fine_tunemodel.h5')



acc = result.history['accuracy']
val_acc = result.history['val_accuracy']
epochs = np.arange(len(acc))


plt.figure()
plt.plot(epochs, acc, label='acc')
plt.plot(epochs, val_acc, label='val_acc')
plt.xlabel('epochs')
plt.ylabel('Accuracy')
plt.title('Learning Curve-finetune')
plt.legend()
```

```
        plt.savefig("finetune_curve.png")
        plt.show()

        from tensorflow.keras.models import load_model
        model = load_model('ownmodel.h5')
        plot_model(model, to_file='model.png')
```

Pair-wise compatibile regression:

Train.py

```
from tensorflow.keras.layers import Dense, concatenate, Input, Conv2D, Flatten,DepthwiseConv2D,MaxPool2D,BatchNormalization
from tensorflow.keras.models import Model
from dataloader import polyvore_dataset
from utils import Config
import tensorflow.keras as tfk
import numpy as np
import tensorflow
import matplotlib.pyplot as plt
from tensorflow.keras.utils import plot_model

if __name__=='__main__':

#         # data generators
        dataset = polyvore_dataset()
        trainList, valList = dataset.readCompat('compatibility_train.txt')
        nClass = 2
        if Config['debug']:
                trainList = trainList[:10000]
                valList = valList[:1500]
        # trainList = trainList[:1]
        # valList = valList[:1]
        trainData = dataset.load(trainList, batchSize=Config['batch_size'])
        valData = dataset.load(valList, batchSize=Config['batch_size'])
        # reg_val = 0.01
        # drop_rate = 0.1
# build model
        x1_input = Input(shape=(224, 224, 3), name='input_image1')
        x1 = Conv2D(32, (3,3),activation='relu')(x1_input)
        x1 = BatchNormalization()(x1)
        x1 = DepthwiseConv2D((3,3))(x1)
        x1 = Conv2D(64, (1,1),activation='relu')(x1)
        x1 = BatchNormalization()(x1)
        x1 = MaxPool2D()(x1)
```

```
x1 = DepthwiseConv2D((3,3))(x1)
x1 = Conv2D(128, (1,1),activation='relu')(x1)
x1 = BatchNormalization()(x1)
x1 = MaxPool2D()(x1)

x1 = DepthwiseConv2D((3,3))(x1)
x1 = Conv2D(256, (1,1),activation='relu')(x1)
x1 = BatchNormalization()(x1)
x1 = MaxPool2D()(x1)

x1 = DepthwiseConv2D((3,3))(x1)
x1 = Conv2D(512, (1,1),activation='relu')(x1)
x1 = BatchNormalization()(x1)
x1 = MaxPool2D()(x1)

first = Flatten()(x1)



x2_input = Input(shape=(224, 224, 3), name='input_image2')
x2 = Conv2D(32, (3,3),activation='relu')(x2_input)
x2 = BatchNormalization()(x2)
x2 = DepthwiseConv2D((3,3), name='dw1')(x2)
x2 = Conv2D(64, (1,1),activation='relu', name='pw1')(x2)
x2 = BatchNormalization()(x2)
x2 = MaxPool2D()(x2)

x2 = DepthwiseConv2D((3,3),name='dw2')(x2)
x2 = Conv2D(128, (1,1),activation='relu', name='pw2')(x2)
x2 = BatchNormalization()(x2)
x2 = MaxPool2D()(x2)

x2 = DepthwiseConv2D((3,3),name='dw3')(x2)
x2 = Conv2D(256, (1,1),activation='relu', name='pw3')(x2)
x2 = BatchNormalization()(x2)
x2 = MaxPool2D()(x2)

x2 = DepthwiseConv2D((3,3),name='dw4')(x2)
x2 = Conv2D(512, (1,1),activation='relu', name='pw4')(x2)
x2 = BatchNormalization()(x2)
x2 = MaxPool2D()(x2)

second = Flatten()(x2)
merge_one = concatenate([first,second], name='merge')
```

```
        # merge = Dense(128, activation='relu')(merge_one)
        merge = Dense(64, activation='relu')(merge_one)
        predictions = Dense(1, activation = 'sigmoid')(merge)
        model = Model(inputs=[x1_input, x2_input], outputs=predictions)
        optimizer = tfk.optimizers.RMSprop(learning_rate=Config['learning_rate'])
        model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
        model.summary()

        # training - num worker is obsolete now
        result = model.fit(trainData, validation_data=valData,
epochs=Config['num_epochs'],shuffle=False)

        model.save('model_compatible.h5')



        acc = result.history['accuracy']
        val_acc = result.history['val_accuracy']
        epochs = np.arange(len(acc))

        plt.figure()
        plt.plot(epochs, acc, label='acc')
        plt.plot(epochs, val_acc, label='val_acc')
        plt.xlabel('epochs')
        plt.ylabel('Accuracy')
        plt.title('Learning Curve-compatible')
        plt.legend()
        plt.savefig("compatible.png")
        plt.show()

        from tensorflow.keras.models import load_model
        model = load_model('model_compatible.h5')
        plot_model(model, to_file='pairwise_model.png')

        dataset = polyvore_dataset(train=False)
        testList, testList2 = dataset.readCompat('compatibility_test_hw.txt')
        testList.append(testList2)
        testList = testList[:100]
        # testList = testList[:2]
        image1, image2 = dataset.load_test(testList)
        temp = model.predict([image1, image2])
        prediction = [int(i>0.5) for i in temp]

        List  = []
        path = os.path.join(self.root_dir, 'train.json')
```

```
    f = open('compatibility_test_hw.txt', 'r')
    for row in f:
        List.append(row[:-1])

    results = np.vstack((List, prediction))
    result = result.T
    f = open('compatible_pred.txt', 'w')
    for row in results:
        temp = row[0]+' ' + row[1] + '\r\n'
        f.write(temp)
```

dataloader.py

```
import tensorflow as tf
import os, json, random
import os.path as osp
from utils import Config

class polyvore_dataset:
    def __init__(self, train=True):
        self.root_dir = Config['root_path']
        self.image_dir = osp.join(self.root_dir, 'images')
        self.Train=train
    '''
    decode one image
    '''
    def decodeImg(self, path):

        image = tf.io.read_file(path)
        image = tf.image.decode_jpeg(image, channels=3)
        image = tf.image.convert_image_dtype(image, tf.float32)

        return image
    '''
    decode file name and return the raw image
    '''
    def process(self, pathAndLabel):
        x = tf.strings.split(pathAndLabel,';')
        image1 = self.decodeImg(x[0])
        image1 = tf.image.resize(image1, (300, 300))
        image1 = tf.image.central_crop(image1, 224 / 300)
        print(tf.shape(image1))
        image2 = self.decodeImg(x[1])
        image2 = tf.image.resize(image2, (300, 300))
        image2 = tf.image.central_crop(image2, 224 / 300)
```

```
        image1 = 2*image1 - 1
        image2 = 2*image2 - 1    # value in [-1,1]
        if self.Train:
            return (image1,image2), tf.strings.to_number(x[2])
        else:
            return image1,image2

    def load_test(self, fileList):
        data = [self.process(path) for path in fileList]
        image1 = [row[0] for row in data]
        image2 = [row[1] for row in data]
        return image1,image2
    '''
    load in data in a streaming fashion
    '''
    def load(self, fileList, batchSize=32):
        data = tf.data.Dataset.from_tensor_slices(fileList)
        # data = data.cache()    #

        data = data.map(self.process, num_parallel_calls=tf.data.experimental.AUTOTUNE)
        data = data.batch(batchSize)
        data = data.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
        return data



    '''
    read in meta info
    '''
    def readCompat(self, filename):
        set_id = {}
        if self.Train:
            meta = open(os.path.join(self.root_dir, 'train.json'), 'r')
        else:
            newpath = '/home/ubuntu/pair/test.json'
            meta = open(newpath, 'r')
            # meta = open(os.path.join(self.root_dir, 'test.json'), 'r')
        meta = json.load(meta)
        for Set in meta:
            item_id = []
            for Items in Set['items']:
                id = Items['item_id']
                item_id.append(id)
            set_id[Set['set_id']] = item_id
```

```
        Compatfile = open(os.path.join(self.root_dir, filename), 'r')
        pair_label = []
        # itemID1, itemID2 = [], []
        for line in Compatfile:
            fields = line.split()
            setID, _ = fields[1].split('_')
            items = set_id[setID]
            if self.Train:
                label = fields[0]
            n = len(items)
            for i in range(n):
                for j in range(i+1,n):
                    if self.Train:

row=osp.join(self.image_dir,items[i])+'.jpg'+';'+osp.join(self.image_dir,items[j])+'.jpg'+';'+lab
el
                        # row=items[i]+';'+items[j]+';'+label
                    else:

row=osp.join(self.image_dir,items[i])+'.jpg'+';'+osp.join(self.image_dir,items[j])+'.jpg'
                        # itemID1.append(items[i])
                        # itemID2.append(items[j])
                    pair_label.append(row)
        if self.Train:
            random.shuffle(pair_label)
        # idx = 1 - int(len(pair_label) * 0.15) # 80/20 split
        idx = 50000
        ending = int(idx*1.15)
        return pair_label[:idx], pair_label[idx:ending]


utils.py
import numpy as np
import os
import os.path as osp
import argparse

Config ={}
# you should replace it with your own root_path
# Config['root_path'] = 'F:\\599dl\\hw4\\polyvore_outfits'
Config['root_path'] = '/home/ubuntu/polyvore_outfits'
# Config['root_path'] = '/root/polyvore_outfits'
Config['meta_file'] = 'polyvore_item_metadata.json'
Config['checkpoint_path'] = ''
```

```
Config['use_cuda'] = True
Config['debug'] = True
Config['num_epochs'] = 8
Config['batch_size'] = 64

Config['learning_rate'] = 0.00001
Config['num_workers'] = 2
```