Major in Computer Science (EC, SO, AT, RM), Major in Electrical and Computer Engineering (SO, AT), Major in Finance (RM), Minor in Mathematics (RM)

An SNN-based Approach to Clothing Image Classification

Erika Cruz, Shota Ogawa, Adithya Thiruvalluvan, Rohit Mahankali

March 7, 2019

425 Brain Inspired Computing

Prof. Konstantinos Michmizos

Abstract

In order to test machine learning methods in the context of vision problems, a clothing image classification system was developed that categorizes the types of clothing given in the image, as well as the color of the clothing. On top of this, the implementation of a recommendation system that would pick similar images to the classified outfit was attempted. A CNN was trained to recognize specific characteristics of inputted images and converted it into an SNN. The work primarily aims to compare the performance when using an SNN versus the same model as a CNN. The model was trained under various labels, which were divided into, as previously suggested, clothing type and color. The CNN model yielded accurate results as long as the input was in the scope of it's labels, and the SNN performed similarly, as long the images were taken in a controlled environment.

1. Introduction

Regardless of whether or not an individual would consider themselves a fashionista or not, the identification and purchasing of clothes is an essential activity in people's lives. Take this premise in tandem with the fact that each individual has their own unique stylistic tastes and choices, and the clothing selection problem becomes quite expansive. And given the advent of AI, and how more and more human-conducted processes are becoming automated, the arrival of intelligent clothing recommendation systems is not far removed. These systems would remove the decision-making fatigue associated with purchasing clothes and given how precise cutting-edge AI is, outputted recommendations would be maximally suitable to the user's style; even to the point that the user would discover clothes that they love, which they previously would not have known would make such a good selection. To contribute to the path, leading to the development of such systems, a basic image classification application for clothes was engineered.

CNN's were the initial machine learning technique used in the study and these networks were converted to SNN's for subsequent processing and computations. This starting point was selected as it is well known in the machine learning community that deep-learning neural networks (DNN's) such as a CNN, have shown great potential as a solution for difficult vision problems, such as object recognition [1]. Beyond for simple objects, these techniques have been found to be suitable for visual learning in general, but these methods fail to integrate the asynchronous, real-time processing capability of the human brain [2]. Given that SNN's have biologically plausible mechanics, useful for asynchronous activity, it is a cutting-edge area of study. To go further, SNN's have been shown to work in architectures that have excellent performance alongside ultra-low power consumption; technology is advancing towards evolutionary neuromorphic hardware [1]. To contribute to this advancement, this study attempts to develop an SNN-based classification system for images of various articles of clothing and try to implement AI logic for user recommendations as well.

2. Background (or Theory)

Although there are many tools today to tackle the problem of visual recognition, there aren't many Nengo models that deal with classification, specifically with clothing classification. By creating the program the goal is seek to diversify and test the NengoDL simulator against

another CNN model the VGG net. In the process, showing the usefulness of machine learning methods in visual recognition [2]. This is to see how the accuracy and performance of the CNN compare to that of the SNN. To that end, the models were tested against both training images and real-life images.

To develop a program for image classification, several resources had to be used as a basis for starting a CNN. This includes many popular deep learning and neural network libraries such as Keras, SKLearn, and VGG net. Keras is an API used to help structure and handle layers of a model it includes several use functions to layer, compile, fit, save, and load.

Through these resources a Convolutional Neural Network was developed that correctly identifies images of clothing. Within this program, the images are classified under variables such as jeans, shirts, and dresses as well as the color of these clothes. SKLearn was used to help with the training process of the models and is especially helpful since creating a multi-label model allows for training of each label. An implementation of VGG net was used as the basis for the CNN, but essentially, it is an image classification CNN that can recognize an image out of a dataset given some training. Using VGG net greatly accelerated the work as it helped to create models that were adept at recognizing certain categories.

Libraries such as Tensorflow, Nengo, and NengoDL were used when converting to an SNN. Tensorflow provided a lot functionality to convert the developed CNN into a Nengo model which allowed the running of NengoDL simulations. This work will show the usefulness of Nengo and its ability to convert CNN's into an SNN. It will also show how the performance of the SNN fares against it's CNN counterpart.

3. Experimental/Modeling Design

The image classification model initially was meant to be a precursor for a recommendation system to find similar clothing images, but due to time constraints the scope ended up decreasing to just a classifier. In order to develop the CNN to use as a base for the SNN, several premade model architectures were looked into to use. Multiple methods were tested separately.

The simplest of these was the 'hello world' of convolutional neural networks the tensorflow neural network tutorial using the Fashion-MNIST dataset. This used a keras sequential model to classify the dataset according to labels after some training. While it had a high accuracy and was easy to use it did not have the specific labels or customizability.

A VGGnet-inspired CNN [3] was what was eventually settled on using for clothing classification using the Keras and SKlearn frameworks. The images were preprocessed by resizing to 96 by 96 pixels, the RGB values were scaled down to 0 to 1, and then flattened to a one-dimensional array. The images were then augmented using the Keras ImageDataGenerator. The batchsize was 32, and the initial learning rate was 0.001. Using the optimizer Adam and loss as binary cross entropy, the model was trained on 75 epochs which took several hours to run on the hardware. Essentially, creating several smaller models, training each to classify their own category and combining them as a multi-label keras model to produce the final CNN using scikit-learn's multi-label binarizer implementation. This VGGNet implementation was then optimized using the keras Adam optimizer. In order to not have to train the model each time to run it as a converted SNN, the model's architecture and weights was saved into a h5 file. A simple user interface was

also created to test the functionality of the CNN and were able to get it to recognize some of the clothings on the person.

The resultant model was transformed into a spiking model using NengoDL, the state-of-the-art framework for developing spiking models. In order to convert from the CNN to an SNN the Nengo 'Inserting a Keras network' [4] tutorial was followed as a basis. Essentially, the keras-based CNN was transformed it several times in order to run it as an SNN. The first being taking the model and making it into a KerasNode and after modifying the class provided by the nengo tutorial the next could be moved onto. The KerasNode was used in order to create a Nengo network which helps to connect the KerasNode model to the inputs and outputs. After this the simulation was all set for displaying of the results. Some issues that were ran into during this conversion process was the fact that the tutorial utilized a CNN based off a version of keras from tensorflow and did not have to load a pre-trained keras model. Another issue was running the simulator while the keras backend was running made led to the realization of needing to clear the keras backend session.

The recommendation model [4], based off a simple resnet found online, was trained using 30000 images from the deepfashion dataset. Only 1 epoch was used. Converting it to an SNN was out of scope. The global pooling layer was used for feature extraction, where each matrix of features was stored into a numpy array. Given a test image, K-nearest neighbor [4] was run against all the features from the training dataset to calculate the similarity between images. The second recommendation method involved resnet-50 with preloaded weights of imagenet, where the average pooling layer was extracted to attempt k-means clustering. It was a struggle to get it to work and when it did it, unfortunately it subjectively produced worse results. A simple cosine similarity based on the same resnet-50 model was also tested separately. The cosine similarity was expected to achieve the same results as k-means clustering if the vectors were normalized, but was not tested on a large dataset.

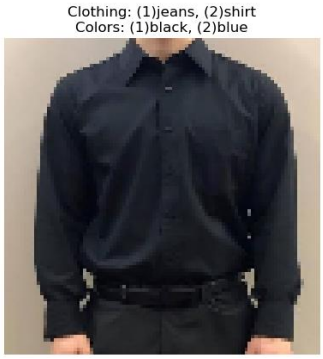
4. Results and Discussion

Classification Discussion:

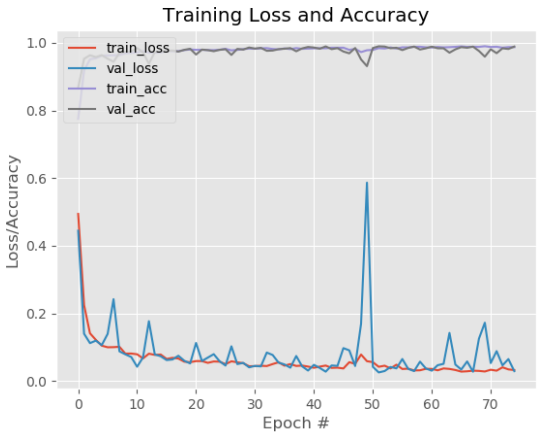
After viewing the accuracy and validation graph, the best time to stop was 25 epochs. At around 50 epochs, the model started to overfit from the sample set, where the validation loss spiked. Similar test images to the training dataset achieved accuracies of around 90%. When converting the model to an SNN using NengoDL, an error occurred for not being able to correctly load the model when called in the post_build function. It was commented out, where it was then built successfully. The predictions were around 50% for all labels; it did not work as expected. The SNN model gave the wrong predictions for every test image, most likely due to not being able to load the weights. Since the nengo-DL guide loaded the model_weights instead of the whole model in post_build, it was previously assumed that loading the weights incorrectly caused this error. After some research done through stack overflow, the model was retrained from scratch, with 25 epochs used, but this time the model_weights were saved. However, the end result was that the model was incorrectly loaded, where a line "with sess.as_default():" before loading it ultimately fixed this error. 40 random images taken from the dataset gave a correct prediction for both color and article of clothing. For custom images, there were minor errors, but the second highest predictions were accurate (however this may be due to the fact that there weren't many labels).

Classification Results:

These are the input images cropped into top and bottom through preprocessing along with the models prediction.



These are the results of training the CNN classification model over 75 and 25 epochs.

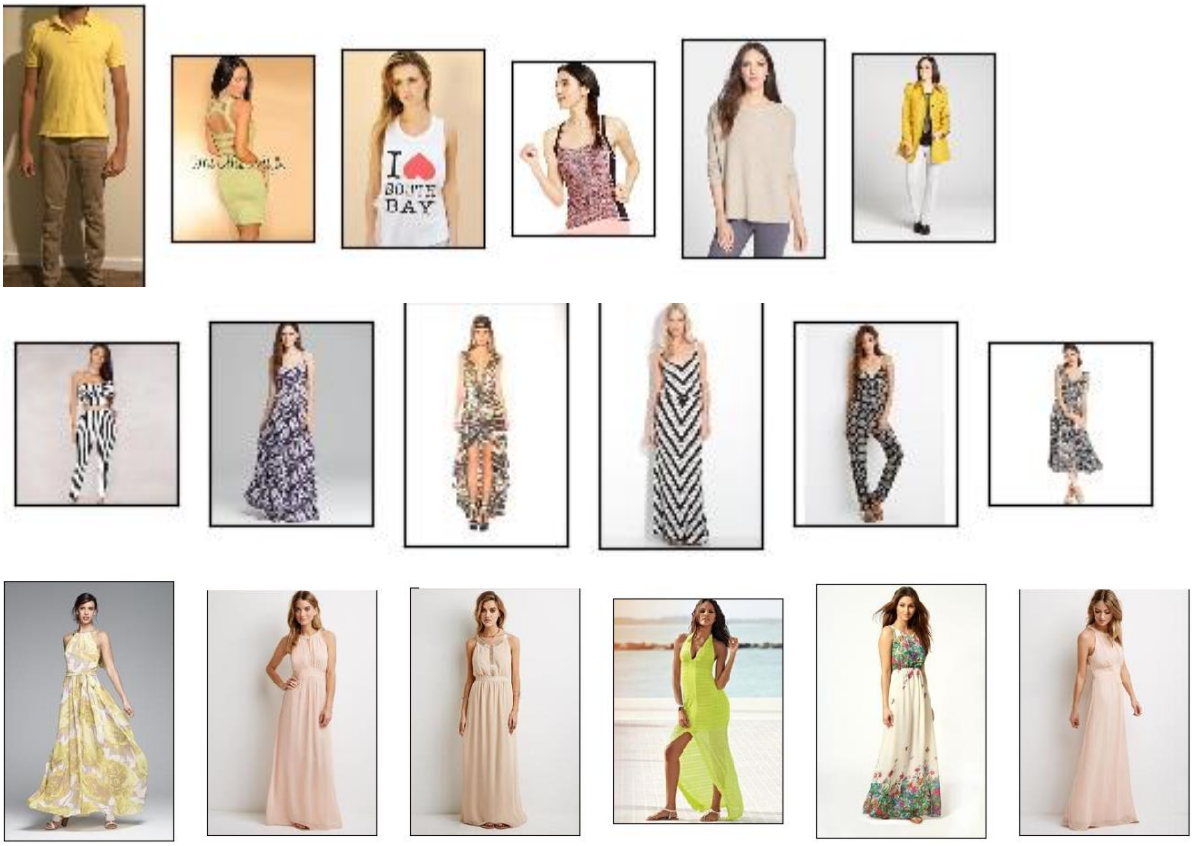


Recommendation Discussion:

The model captured the attributes of the test images that were split from the overall dataset fairly well; the recommended images were subjectively similar (no actual validation was done). The actual predictions for this model were incorrect and random, therefore it was omitted from results. For the images that were personally taken, the recommendations produced captured a few attributes, but was not as close as the test dataset. Given that the training and test dataset were chosen from random, the results were satisfactory. The future direction and the more optimal solution is to develop a custom dataset for specific recommendations, such as dress clothing, casual wear, athletic wear, etc, instead of randomly choosing from the DeepFashion dataset. The conversion from the simple resnet to an SNN produced multiple errors that could not be well understood. Due to time constraints, it was left unconverted.

Recommendation Results:

These are the input image (left) and 5 similar output images (right) for the recommendation CNN.



5. Conclusions

At the end of the study, an SNN was developed that was able to classify clothing images with accuracy. The training of the CNN with the image classification had ended with a 90% accuracy rate - much in agreement with the CNN accuracy discussed in the 2014 study by Cao and Chen [5]. Upon conversion to the SNN paradigm, similarly accurate results were discovered. A more in depth-analysis is required to assess the extent of performance differences between these two model architectures. A working clothing recommendation system was also partially developed but conversion to an SNN was infeasible due to time constraints. In regards to the recommendation system, the extent of NengoDL library was explored and a collection of different neural networks were prepared for study; conclusions proclaiming superior performance in SNN's could not be drawn from the scope of this work.

Acknowledgments

The authors would like to thank Prof. Konstantinos Michmizos for providing the instruction in completing this study as well as providing guidelines for this report. The authors

would also like to thank Neelesh Kumar and Guangzhi Tang for providing their support and steering us in the right direction with their advice.

References

[1] Cao, Y., Chen, Y., and Khosta, D., 2015, “Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition.”, International Journal of Computer Vision 113.1, pp. 54-66.

[2] Tang, G. and Sun, Z., 2015, “A Biologically Plausible Model for Human Visual Perception on Handwriting Digits.”

[3] “Multi-label classification with Keras,” *PyImageSearch*, 27-May-2018. [Online]. Available: https://www.pyimagesearch.com/2018/05/07/multi-label-classification-with-keras. [Accessed: 08-May-2019].

[4] “Inserting a TensorFlow/Keras network into a Nengo model,” Inserting a TensorFlow/Keras network into a Nengo model - NengoDL documentation. [Online]. Available: https://www.nengo.ai/nengo-dl/examples/tensorflow-models.html. [Accessed: 08-May-2019].

[5] “Deep Shopping: Content-based Clothing Image Retrieval Using Deep Features,” *Wenxin’s Deep Learning Blog*, 11-Jan-2017. [Online]. Available: https://wenxinxublog.wordpress.com/2017/01/06/deep-shopping. [Accessed: 08-May-2019].

Appendix

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import nengo
import nengo_dl

from tensorflow.python.keras.models import load_model, clone_model
from tensorflow.python.keras import backend as K
from tensorflow.python.keras.preprocessing.image import img_to_array
import glob
import pickle
import h5py


def SNN_Clothing_Classification():
  """
  K.clear_session()
  data = []
```

```
  labels = [['black', 'jeans']]*344 + [['blue', 'dress']]*386 + [['blue', 'jeans']]*356 + [['blue',
'shirt']]*369 + \
         [['blue', 'sweater']]*99 + [['gray', 'shorts']]*96 + [['red', 'dress']]*380 + [['red',
'shirt']]*332

  EPOCHS = 25
  INIT_LR = 1e-3
  BS = 32
  IMAGE_DIMS = (96, 96, 3)



  checkpoints_dir = '.\\checkpoints'
  # load the image, pre-process it, and store it in the data list
  import glob
  image_types = ('*.jpg', '*.jpeg', '*.png')
  files = []
  labels = []
  for image_type in image_types:

files.extend(glob.glob("D:\\Users\\bob\\PycharmProjects\\test_recommendation\\nengo_classif
ication\\test_images\\" + image_type))
  print(len(files))
  for f in files:
      image = cv2.imread(f)
      image = cv2.resize(image, (IMAGE_DIMS[1], IMAGE_DIMS[0]))
      image = img_to_array(image)
      data.append(image)
      label = f.split("\\")[-2].split("_")
      labels.append(label)

  # scale the raw pixel intensities to the range [0, 1]
  data = np.load("data.npy")
  labels = np.array(labels)

  # print("[INFO] data matrix: {} images ({:.2f}MB)".format(len(imagePaths), data.nbytes /
(1024 * 1000.0)))

  # binarize the labels using scikit-learn's special multi-label
```

```
# binarizer implementation
# print("[INFO] class labels:")
mlb = MultiLabelBinarizer()
labels = mlb.fit_transform(labels)

#print(labels)

# loop over each of the possible class labels and show them
for (i, label) in enumerate(mlb.classes_):
    print("{}. {}".format(i + 1, label))

print(data.shape)
print(labels.shape)
# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data,
                                    labels, test_size=0.2, random_state=42)

# construct the image generator for data augmentation
aug = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,
                    height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
                    horizontal_flip=True, fill_mode="nearest")

# initialize the model using a sigmoid activation as the final layer
# in the network so we can perform multi-label classification
print("[INFO] compiling model...")
model = SmallerVGGNet.build(
    width=IMAGE_DIMS[1], height=IMAGE_DIMS[0],
    depth=IMAGE_DIMS[2], classes=len(mlb.classes_),
    finalAct="sigmoid")

# initialize the optimizer (SGD is sufficient)
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)

# compile the model using binary cross-entropy rather than
# categorical cross-entropy -- this may seem counterintuitive for
# multi-label classification, but keep in mind that the goal here
# is to treat each output label as an independent Bernoulli
# distribution
model.compile(loss="binary_crossentropy", optimizer=opt,
```

```
            metrics=["accuracy"])

print("[INFO] training network...")
H = model.fit_generator(
    aug.flow(trainX, trainY, batch_size=BS),
    validation_data=(testX, testY),
    steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS, verbose=1)

# save the model to disk
print("[INFO] serializing network...")
model.save("fashion_model.h5")
model.save_weights("fashion_model_weights.h5")

plt.style.use("ggplot")
plt.figure()
N = EPOCHS
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["acc"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="upper left")
plt.savefig("plot.png")
"""
IMAGE_DIMS = (96, 96, 3)
K.clear_session()
class KerasNode:
    def __init__(self, keras_model, mlb):
        self.model = keras_model
        self.mlb = mlb

    def pre_build(self, *args):
        self.model = clone_model(self.model)

    def __call__(self, t, x):
        # pre-process the image for classification
        img = tf.reshape(x, (-1,) + IMAGE_DIMS)
```

```
        return self.model.call(img)

    def post_build(self, sess, rng):
        with sess.as_default():
            self.model.load_weights("fashion_model_weights.h5")
            self.mlb = pickle.loads(open("mlb.pickle", "rb").read())
        #pass

net_input_shape = np.prod(IMAGE_DIMS)  # because input will be a vector

with nengo.Network() as net:
    # create a normal input node to feed in our test image.
    # the `np.ones` array is a placeholder, these
    # values will be replaced with the Fashion MNIST images
    # when we run the Simulator.
    input_node = nengo.Node(output=np.ones((net_input_shape,)))

    # create a TensorNode containing the KerasNode we defined
    # above, passing it the Keras model we created.
    # we also need to specify size_in (the dimensionality of
    # our input vectors, the flattened images) and size_out (the number
    # of classification classes output by the keras network)
    model = load_model("fashion_model.h5")
    mlb = pickle.loads(open("mlb.pickle", "rb").read())
    keras_node = nengo_dl.TensorNode(
        KerasNode(model, mlb),
        size_in=net_input_shape,
        size_out=len(mlb.classes_))

    # connect up our input to our keras node
    nengo.Connection(input_node, keras_node, synapse=None)

    # add a probes to collect output of keras node
    keras_p = nengo.Probe(keras_node)
    input_p = nengo.Probe(input_node)


files = []
image_types = ('*.jpg', '*.jpeg', '*.png')
test_images_path = ".\\test_images2\\"
```

```
for image_type in image_types:
    files.extend(glob.glob(test_images_path + image_type))

data = []
for file in files:
    image = cv2.imread(file)
    image = cv2.resize(image, (IMAGE_DIMS[1], IMAGE_DIMS[0]))
    image = img_to_array(image)
    data.append(image)

data = np.array(data, dtype="float") / 255.0
minibatch_size = len(data)
print(minibatch_size)
# flatten images so we can pass them as vectors to the input node
test_inputs = data.reshape((-1, net_input_shape))

# unlike in Keras, NengoDl simulations always run over time.
# so we need to add the time dimension to our data (even though
# in this case we'll just run for a single timestep).
test_inputs = test_inputs[:, None, :]

with nengo_dl.Simulator(net, minibatch_size=minibatch_size) as sim:
    sim.step(data={input_node: test_inputs})

tensornode_output = sim.data[keras_p]

clothing = ["jeans", "dress", "shirt", "sweater", "shorts"]
colors = ["blue", "black", "gray", "red"]

for i in range(minibatch_size):
    plt.figure()
    b, g, r = cv2.split(data[i])
    rgb_img = cv2.merge([r, g, b])
    plt.imshow(rgb_img)
    probs = tensornode_output[i][0]
    print(probs)
    highs = np.argsort(probs)[::-1]
    #search for two highest clothing and colors
    cloth = []
    cloth_c = 0
```

```
        color = []
        colors_c = 0
        for i in highs:
            if cloth_c < 2:
                if mlb.classes_[i] in clothing:
                    cloth.append(i)
                    cloth_c += 1
            if colors_c < 2:
                if mlb.classes_[i] in colors:
                    color.append(i)
                    colors_c += 1

        plt.axis("off")
        plt.title("Clothing: (1){}, (2){}\nColors: (1){}, (2){}".format(mlb.classes_[cloth[0]],
mlb.classes_[cloth[1]], mlb.classes_[color[0]], mlb.classes_[color[1]]))
        plt.show()

    K.clear_session()

SNN_Clothing_Classification()
```

```
import sys
import cv2
import matplotlib.pyplot as plt

import tensorflow as tf
import pandas as pd
import numpy as np
import glob

import KNN_test2.fashion_input as fi
import KNN_test2.simple_resnet as sr
IMG_ROWS = fi.IMG_ROWS
IMG_COLS = fi.IMG_COLS

class KNN:
    def __init__(self):
```

```
        self.placeholders()

    def loss(self, logits, bbox, labels, bbox_labels):
        labels = tf.cast(labels, tf.int64)
        cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=logits,
labels=labels,
                                                                       name='cross_entropy_per_example')
        mse_loss = tf.reduce_mean(tf.losses.mean_squared_error(bbox_labels, bbox),
name='mean_square_loss')
        cross_entropy_mean = tf.reduce_mean(cross_entropy, name='cross_entropy')
        return cross_entropy_mean + mse_loss

    def top_k_error(self, predictions, labels, k):
        batch_size = predictions.get_shape().as_list()[0]
        in_top1 = tf.to_float(tf.nn.in_top_k(predictions, labels, k=k))
        num_correct = tf.reduce_sum(in_top1)
        return (batch_size - num_correct) / float(batch_size)

    def placeholders(self):
        self.lr_placeholder = tf.placeholder(dtype=tf.float32, shape=[])
        self.dropout_prob_placeholder = tf.placeholder(dtype=tf.float32, shape=[])

    def test_single(self, num_images):
        self.test_image_placeholder = tf.placeholder(dtype=tf.float32, shape=[num_images,
IMG_ROWS,
                                                               IMG_COLS, 3])
        self.test_label_placeholder = tf.placeholder(dtype=tf.int32, shape=[num_images])

        ############################
        # Build test graph
        logits, bbox, global_pool = sr.inference(self.test_image_placeholder, n=2, reuse=False,
                                         keep_prob_placeholder=self.dropout_prob_placeholder)
        predictions = tf.nn.softmax(logits)
        test_error = self.top_k_error(predictions, self.test_label_placeholder, 1)

        saver = tf.train.Saver(tf.global_variables())
        init = tf.global_variables_initializer()
        sess = tf.Session()
        sess.run(init)
        saver.restore(sess, 'model.ckpt-30000')
```

```
        print('Model restored!')
        ############################

        test_df = fi.prepare_df(".\\labels_modified_demo2.csv",
                                usecols=['image_name', 'category', 'x1_modified', 'y1_modified',
'x2_modified',
                                         'y2_modified'], shuffle=False)

        prediction_np = np.array([]).reshape(-1, sr.NUM_LABELS)

        fc_np = np.array([]).reshape(-1, 64)

        # Hack here: 25 as batch size. 50000 images in total
        df_batch = test_df.iloc[0: len(test_df), :]
        test_batch, test_label, bbox_array = fi.load_data_numpy2(df_batch)

        prediction_batch_value, test_error_value = sess.run([predictions, test_error], feed_dict={
            self.test_image_placeholder: test_batch, self.test_label_placeholder: test_label})
        fc_batch_value = sess.run(global_pool, feed_dict={
            self.test_image_placeholder: test_batch, self.test_label_placeholder: test_label})

        prediction_np = np.concatenate((prediction_np, prediction_batch_value), axis=0)
        fc_np = np.concatenate((fc_np, fc_batch_value))

        print('Predicting array has shape ', fc_np.shape)
        np.save(".\\test_fc_demo2", fc_np)
        np.save('.\\test_prediction_demo2', prediction_np)

    def preprocess_from_images_folder(self):
        data_folder =
"D:/Users/bob/PycharmProjects/test_recommendation/KNN_test2/labels_modified_demo2.csv
"
        df = pd.read_csv(data_folder,
                         sep=',',
                         header=0)  # ,
        # names=['image_name', 'category_type', 'category', 'x1', 'y1', 'x2', 'y2'])
        j = 0
        for i, image in df.iterrows():
            if i % 1000 == 0:
                print(i)
```

```
            name = image['image_name']
            img = cv2.imread(name)
            if img is None:
                continue
            h = img.shape[0]
            w = img.shape[1]
            df.loc[i, 'x1_modified'] = float(df.loc[i, 'x1']) / float(w)
            df.loc[i, 'y1_modified'] = float(df.loc[i, 'y1']) / float(w)
            df.loc[i, 'x2_modified'] = float(df.loc[i, 'x2']) / float(h)
            df.loc[i, 'y2_modified'] = float(df.loc[i, 'y2']) / float(h)
            j += 1

df.to_csv(r'D:/Users/bob/PycharmProjects/test_recommendation/KNN_test2/labels_modified_
demo2.csv', index=False)
        return j

test = KNN()
#images_len = test.preprocess_from_images_folder()
#test.test_single(images_len)
category_path = ".\\Category and Attribute Prediction
Benchmark\\Anno\\list_category_cloth.txt"
try:
  category_file = open(category_path, 'r')
except Exception:
  print('error opening category_cloth_file:', category_path)
categories = []
lines = category_file.readlines()
for line in lines[2:]:
  category, category_type = line.split()
  categories.append(category)

test_set = np.load('.\\30000_references.npy')
input_set = np.load(".\\test_features.npy")
predictions = np.load('.\\test_prediction.npy')

from sklearn.neighbors import NearestNeighbors

neighbors = NearestNeighbors(n_neighbors=5, algorithm='auto').fit(test_set)

input_csv = ".\\test_modified.csv"
```

```
input_df = pd.read_csv(input_csv, usecols=['image_name'])

output_path = ".\\Category and Attribute Prediction Benchmark\\Img\\"
output_csv = ".\\train_modified.csv"
output_df = pd.read_csv(output_csv, usecols=['image_name'])


for i in range(len(input_set)):
  fig = plt.figure()
  image_name = input_df.loc[i, 'image_name']
  #fig.suptitle("{}".format(image_name))
  fig.suptitle("{}, prediction: {}".format(image_name,
categories[int(np.argmax(predictions[i]))]))
  distances, indices = neighbors.kneighbors([input_set[i]])
  plt.subplot(1, 6, 1)
  img = cv2.imread(output_path + image_name)
  b2, g2, r2 = cv2.split(img)
  rgb_img2 = cv2.merge([r2, g2, b2])
  plt.imshow(rgb_img2)
  plt.xticks([]), plt.yticks([])
  j = 2
  for index in indices[0]:
      plt.subplot(1, 6, j)
      subpath = output_df.loc[index, 'image_name']
      img = cv2.imread(output_path + subpath)
      b2, g2, r2 = cv2.split(img)
      rgb_img2 = cv2.merge([r2, g2, b2])
      plt.imshow(rgb_img2)
      plt.xticks([]), plt.yticks([])
      j += 1
  plt.show()
```