REVERSE IMAGE SEARCH

Authors

Anil Poonai (ap5254@nyu.edu**)**
Sakshat Katyarmal (sk9428@nyu.edu**)**
Utsav Patel (up2023@nyu.edu**)**

ABSTRACT

In this project we aim to create a Reverse Visual Search Engine. The objective of the reverse visual search is to find similar images to an image of our choice that contains a person of interest (POI) from the Labeled Faces in the Wild(LFW) dataset. Each POI used has exactly 6 matching images of themselves in the dataset. For this project we have made 2 reverse image search models: the first one being the baseline model and the second one being an improvement model. The baseline is supposed to be a rough neural network built from scratch, while the improvement is supposed to be a high performing pretrained model. At the end of this project a user will be able to select an image of their choice and our model will give the user an output of 20 images, including the original query image, which the model finds the most similar to the query image. All the models were created using tensorflow.

INTRODUCTION

We wanted our image search engine to be able to match images to each pixel, we specifically wanted it to match faces. This is important in multiple realms including identifying missing persons and criminals, access rights(phone, system, ride sharing), medical treatments, etc. Our results show that this is reachable and that while improvement needs to be done is it possible to get a reliable image searching algorithm without cutting edge technology.

RELATED WORK

For a reference to research, look no further than Yann LeCun's paper "Object Recognition with Gradient Based Learning". His approach to digit recognition of the MNIST dataset, Lecun 5, is similar to our baseline approach using only convolution and pooling layers totaling 7 layers in total. Ours differ in the number of layers and filters we use in each convolution, but the architecture is similar. Regarding our improvement, you can find the pretrained network itself documented in the article "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning'', it goes into more depth into their architecture although we talk about it as well as the techniques it uses as well.

DATA

For this project we'll be using the Labeled Faces in the Wild (LFW) dataset. This is a dataset of more than 13,000 images of people labeled with their names. From these 13,000 images, 1680 images of the people have two or more distinct photos in the dataset. In total there are 5749 people in these images.We got the dataset from tensorflow datasets. The queries we are using from this dataset all have 6 images of themselves in the dataset.

PREPROCESSING

We converted the data from a tensorflow tensor to a numpy array just so we could visualize it easier. This was easy for the images to convert but it left the labels with a 'b' in front of it. This wasn't a big deal as we made all of the images into a file called imags.npy, the labels into a file called labellistarr.npy and the numeric classification of the labels for the tensorflow model into a file called, endlabels.npy. We also saved the baseline mode into a folder called ModelReal and processed the features of the images through the model as features.npy, which is the only file needed for the image similarity search through the K- nearest neighbor algorithm. We then did the same thing with the pretrained model and labeled the features as impovfeatures.npy.

METHODS

Architecture

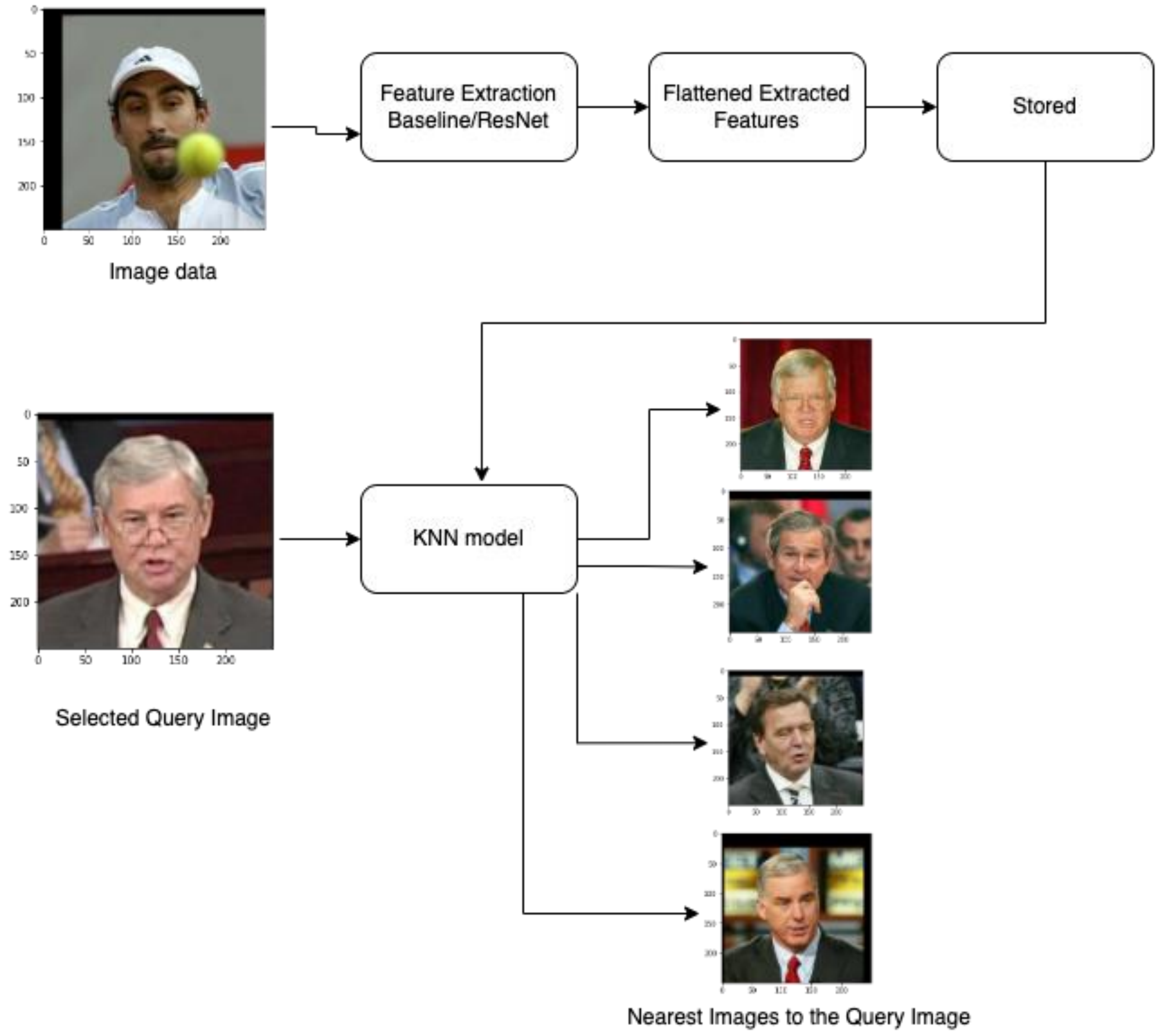


Image data
Feature Extraction Baseline/ResNet
Flattened Extracted Features
Stored
KNN model
Selected Query Image
Nearest Images to the Query Image

As you can see above we took the dataset and after building the baseline CNN model using the LFW dataset and getting the pretrained model with weights from IMageNet, we ran the dataset through each model again but without the fully connected layer so we would have the features extracted in a vector instead of as a classified image. We then stored them into the features and improvfeatures files mentioned earlier. After this part was finished, we created the nearest neighbor classifications by having 20 neighbors(20 images grouped together), using the Brute-Force method since it’s the most accurate as it used every data point and send them towards the distance metric but is relatively slow compared to KD Tree and Ball Tree as well as the minkowski method to calculate distance which is a generalized method as opposed to a method like manhattan distance which works well in a grid like system.

Baseline Model

Feature Extraction

Dataloading- For the Baseline model, we faced issues as we were constantly running out of ram as there were a lot of variables in our output layer. To test that out we limit our output prediction layer to the first 250 unique labels. After we confirmed the limits we used two solutions, one of which had an RTX 3060 GPU which we used for the model processing, but even this failed so we went with our second solution, using Collab Pro, which although slow, did manage to give results. We managed to run the models and save the files mentioned previously in the Preprocessing section for easier use.

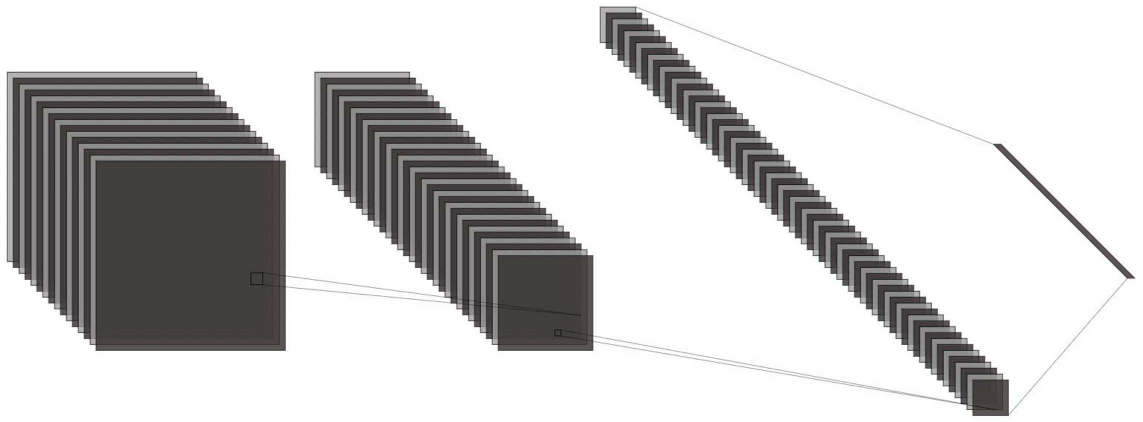
Model & Training- For the purpose of Baseline we made a simple CNN Model using Tensorflow

```
 Sequential [
   Conv2D(32, (3,3), padding='valid', activation="relu", strides=1, input_shape=(250,
250, 3)),
    MaxPooling2D((2, 2), strides=2),

    Conv2D(64, (3,3), padding='valid', activation="relu"),
    MaxPooling2D((2, 2), strides=2),

    Conv2D(128, (3,3), padding='valid', activation="relu"),
    MaxPooling2D((2, 2), strides=2),

    Flatten(),
    Dense(5749, activation="softmax")
]
```



Simple Architecture Diagram for our baseline Feature Extraction model 32, 64 and 128 layers with MaxPool layers after each convolution layer. The last layer is the flattened layer with all the features.

Here we have used relu and softmax activation functions with 3 Convolutional layers in which kernel size is 3x3 , stride is 1 and we are feeding 250x250 RGB image size to the network altogether with a maxpool layer with each convolution layer .

To train our model we have used "sparse_categorical_crossentropy" as our loss function and adam optimizer. We train our model for 10 epochs with a batch size of 100. With this we were able to achieve an accuracy of 96%.

Implementation

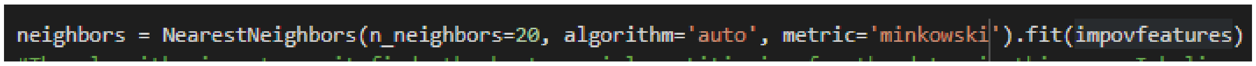
For reverse image search implementation we would not need the output layer or the classification layer. Our main aim here is only the feature extraction. To do this we remove the last 2 layers of our Baseline model, that way only features are extracted from the images. We save these flattened features.

KNN model- To group similar images we use the K nearest neighbor model.The algorithm is auto so it finds the best spatial partitioning for the data (brute force), so there aren't any weights attached to the spatial divide between the features. The metric we are using is minkowski, which is the distance method between the features.

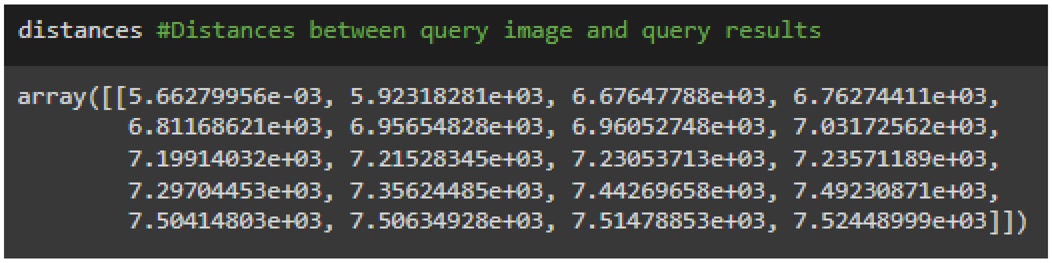
NearestNeighbors(n_neighbors=20, algorithm='auto', metric='minkowski').fit(features)

distances, indices = neighbors.kneighbors([features[696]])

From this we are trying to find 20 images which are nearest to the image at index 696. The variable ‘**distances**’ stores an array of distances between the 5 images and the query images. The indices variable stores an array of the indexes of the images which are nearest to the query image. For eg-



```
neighbors = NearestNeighbors(n_neighbors=20, algorithm='auto', metric='minkowski').fit(impovfeatures)
```



```
distances #Distances between query image and query results

array([[5.66279956e-03, 5.92318281e+03, 6.67647788e+03, 6.76274411e+03,
        6.81168621e+03, 6.95654828e+03, 6.96052748e+03, 7.03172562e+03,
        7.19914032e+03, 7.21528345e+03, 7.23053713e+03, 7.23571189e+03,
        7.29704453e+03, 7.35624485e+03, 7.44269658e+03, 7.49230871e+03,
        7.50414803e+03, 7.50634928e+03, 7.51478853e+03, 7.52448999e+03]])
```

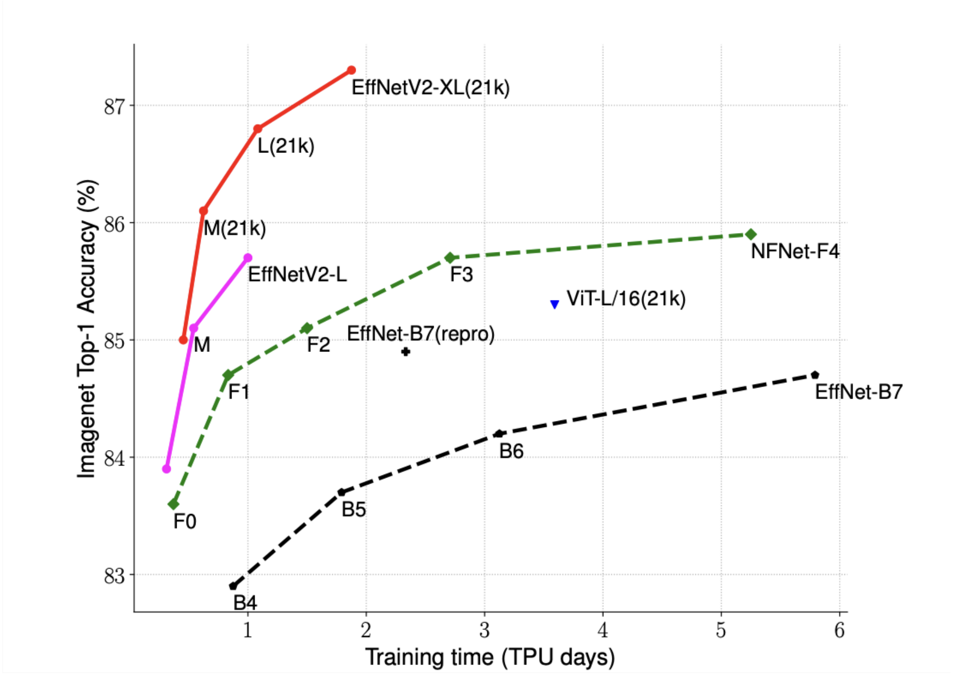
As you can see from the above output the minkowski distance metrics shows the nearest 20 images to the image we selected or queried.Our aim for the Improvement model will be to be able to reduce the distance between the queried images and nearest found images. We elaborate on what features in the images those distances represent in the EXPERIMENTS & IMPROVEMENTS section below.

Improvement Model

EfficientNet V2L

Here we are using the EfficientNet V2L which has better efficiency and fast training speed. This model is even smaller than the other models yet it gives better results. This model will give us a training efficiency of around 84%.

model = tf.keras.applications.EfficientNetV2L(weights='imagenet', include_top=False, input_shape=(250, 250, 3))



Training Efficiency

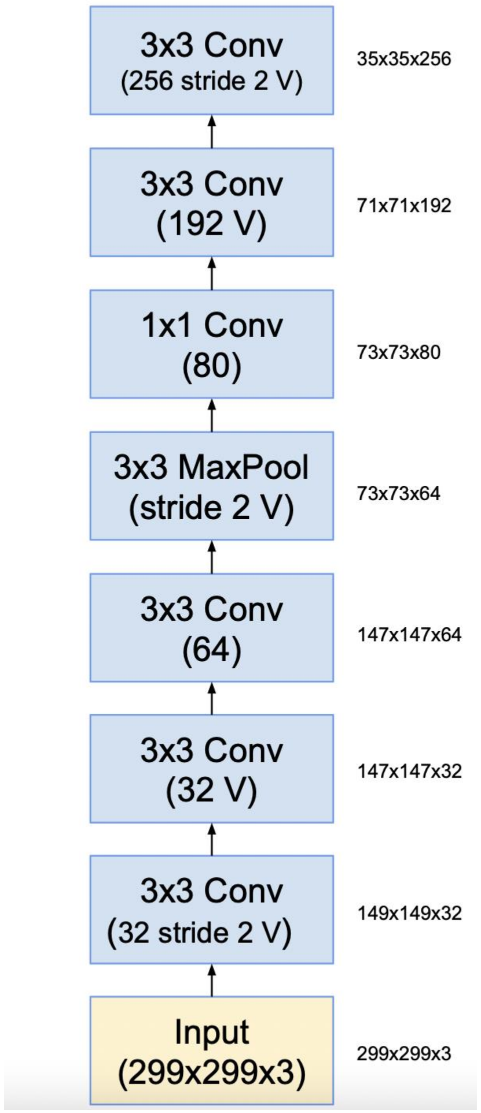
The weights= ‘imagenet’ means we are pre-training on ImageNet. The inclue_top parameter means whether we include a fully connected (FC) layer on the top of the network or not. In this case we have not included the FC layer on the top. The input_shape should have 3 input channels which in this case is 250,250.

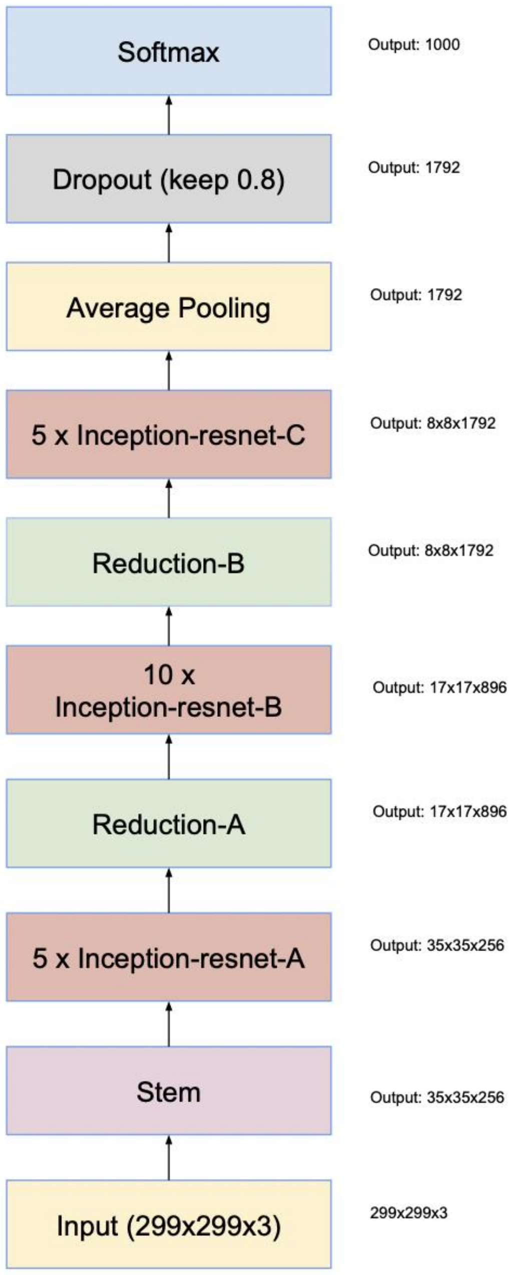
ResNet V2

We’ll also be using the Inception Resnet V2 as it gives better performance. It has been proven that when an inception block is combined with residual connections it tends to perform slightly better than only an inception block. For scaling up the dimensionality the inception block is followed by a filter-expansion layer. Also in the residual inception network we use batch normalization on the top of traditional layers. Below is the model we used:

```
model = tf.keras.applications.InceptionResNetV2(weights='imagenet',
include_top=False, input_shape=(250, 250, 3))
```

The weights= ‘imagenet’ means we are pre-training on ImageNet. The inclue_top parameter means whether we include a fully connected (FC) layer on the top of the network or not. In this case we have not included the FC layer on the top. The input_shape is the width and height of the matrix which in this case is 250,250. Also in the input_shape the height and width should be more than 75.





A) Inception ResNet V2 size **B) Inception ResNet V2 architecture**

In the above given block diagram we can take the input of size (299x299x3) but in our case we'll be taking the input (250x250x3). For feature extraction we don't require average polling and softmax layers.

EXPERIMENTS & IMPROVEMENTS

Regarding our experimentation, let's focus on the baseline first. So, our baseline doesn't match faces well but it seems to match outfits(suits, ties) and hairstyles(Including facial hair). As we go through the images, we notice that the person of interest themselves isn't as prominent in the k-nearest neighbors algorithm, but the attributes mentioned above and the background and position are prominent. In our improvement, we used the InceptionResNetV2 model, this did improve our search engine but not in the way we originally wanted. In the baseline, the person of interest would be matched with people of different genders and different positions in the photo, with our improvement model, it didn't mismatch as many genders. Also, in all of the photos of the baseline there were a lot of the same photos that kep cycling through each search, four of the photos are of Paris Hilton, Nicole, Matthew Perry and Christopher Russell were prominent in multiple queries and it looks like it's due to the generic position of themselves in the frame and the background that has a similar pattern to a lot of other images of being a lighter color and more consistent through the image. They are shown below:

Nicole

Paris Hilton

While in our improvement model we don’t have that problem. It also improved on all of the characteristics mentioned above that the first model did well on but still failed on facial recognition, although it did mismatch race less as well. The improvement model isn’t perfect but it does show what a deeper network can accomplish if allotted more time and compute power. I believe the biggest change was the use of inception blocks as they do a deeper dive into image characteristics due to the use of multiple convolutions of different sizes on the same input instead of a pure linear single convolution being used. This allows for more data to be extracted and transformed into vectors.

CONCLUSION

Through our implementation of our own model and our implementation of pretrained models, we’ve determined that, as it is possible to create a facial recognition algorithm but there are a lot of restrictions such as RAM and computing power, this is due to facial recognition taking up such a small amount of the pixels compared to the overall image. It is highly doable with enough time and power but I believe a deeper neural network is better with methods such as an inception layer being used. While at least one of our models did an excellent job at matching similar images regarding position in frame, gender, hair, attire, background, etc. Facial features themselves weren't great, I did see a noticeable improvement from the baseline to the improvement model as the faces did look more alike especially regarding gender, race and even age, it didn’t really match the same person. We learned a lot about the type of features that are noticed by a neural network and the ways that we can improve them, we also learned about the restrictions and expenses of them as well. If we had the compute power I would’ve changed the improvement models to something more along the lines of facenet while the baseline would’ve had less layers but more deep layers, as that seems to do better at finding differences.

Project can be found here: **https://github.com/DevonARP/NYU_AI_6613**
Differences in the photos are shown in the repo in order to see the model outputs.

RELATED RESEARCH & REFERENCES

1. *Szegedy, Christian, et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." Thirty-first AAAI conference on artificial intelligence. 2017.*
2. *LeCun, Yann et al. "Object Recognition with Gradient-Based Learning." Shape, Contour and Grouping in Computer Vision (1999).*
3. *Koul, A., Ganju, S., &amp; Kasam, M. (n.d.). Practical deep learning for cloud, mobile, and edge. O'Reilly Online Learning. Retrieved April 30, 2022, from https://www.oreilly.com/library/view/practical-deep-learning/9781492034858/ch04.html*
4. *Yinleon. (n.d.). Yinleon/Pydata2017: This is the code and presentation for my PYDATA2017 talk "reverse image search using out-of-the-box machine learning libraries. GitHub. Retrieved April 30, 2022, from https://github.com/yinleon/pydata2017*