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 the image which we select. For this project we have made 2 reverse image search models- first one is the baseline model and the second one is an improvement model. At the end of this project a user will be able to select an image of his choice and our model will be giving the user an output of 5 images which are similar to the one the user selects. All the testcases are implemented using Deep Learning techniques like tensorflow, keras, scikit learn.

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.https://github.com/yinleon/pydata2017/blob/master/pydata-yin-reverse_img_search.ipynb

3.https://www.oreilly.com/library/view/practical-deep-learning/9781492034858/ch04.html

<u>ARCHITECTURE AND INTRODUCTION</u>

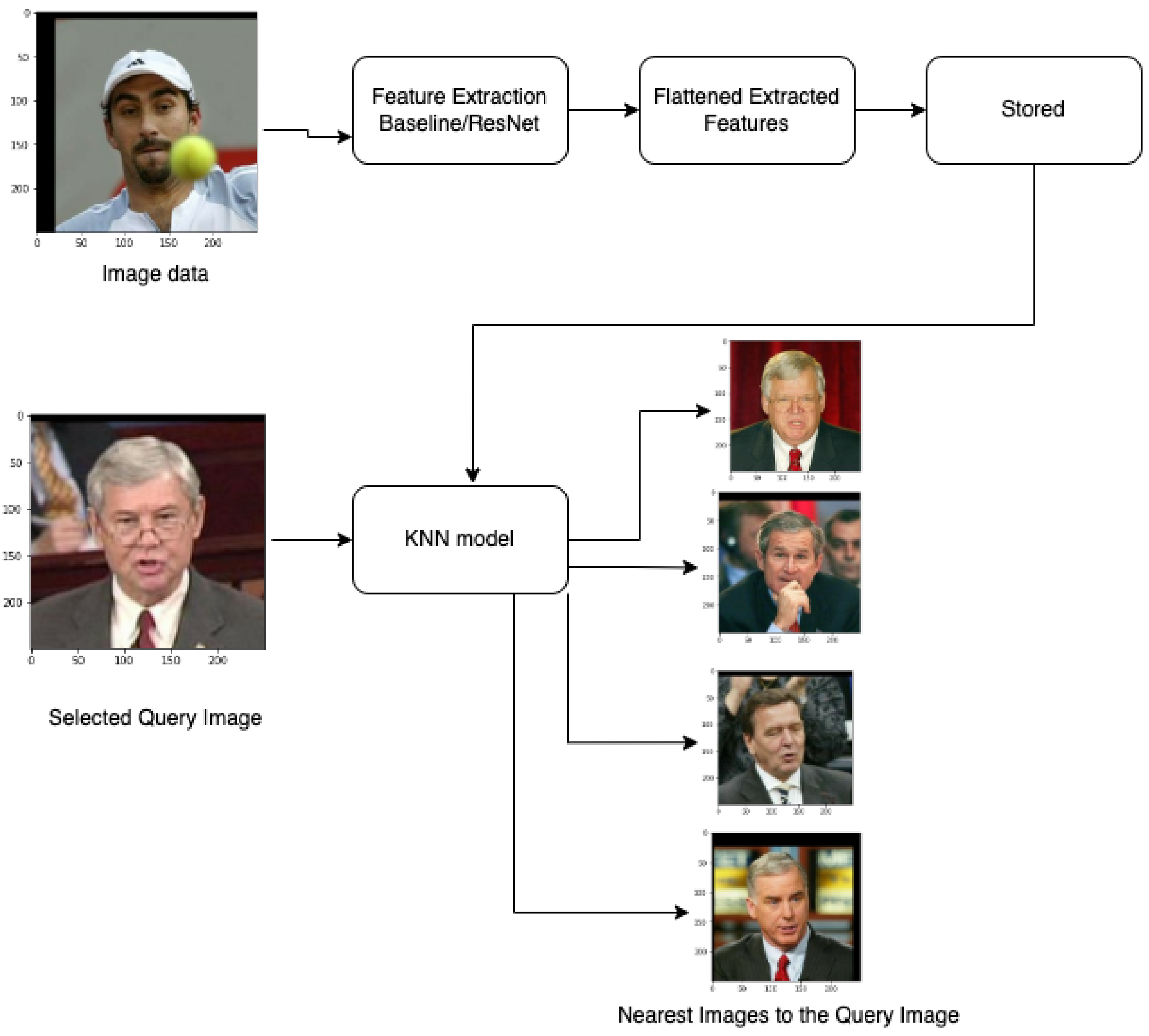


Image data
Feature Extraction Baseline/ResNet
Flattened Extracted Features
Stored
Selected Query Image
KNN model
Nearest Images to the Query Image

This architecture shows how we worked on our implementation of the reverse visual search algorithm. In a nutshell First we process our data, extract labels and images and make loaders for these. We feed our loaders for feature extraction to our Baseline or ResNet model/ Improvement Model. These Features are then stored in a flattened np array and we save it in our local system. This flattened features array is then fed to a KNN Model which eventually gives us 5 images which are very similar to our image which we select.

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 are loading these using tensorflow.

Preprocessing

We then preprocess the data in such a way that we have labels stored in a numpy array. We convert the categorical elements of the numpy array to numerical values such that every "Name" corresponds to a number. Similarly we are storing all the images in a separate array. In this way we make our initial loaders.

METHODS

Baseline Model

Feature Extraction

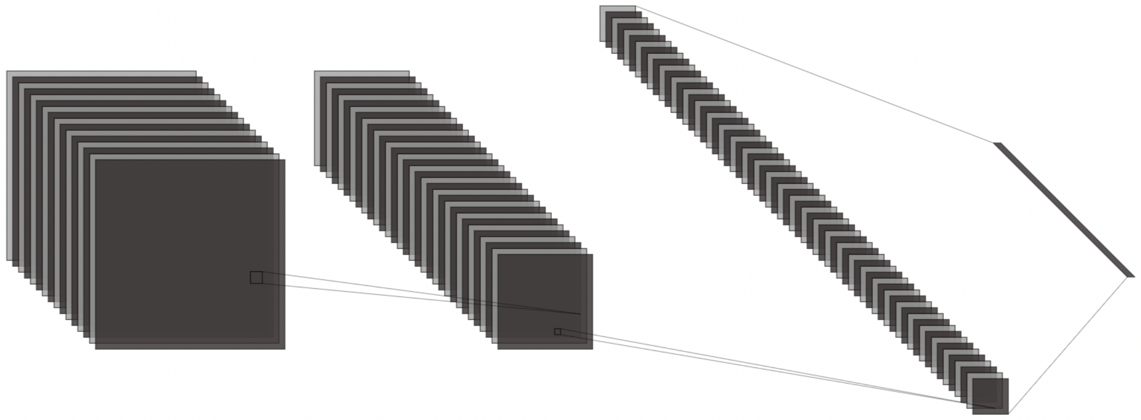
Dataloading-For the Baseline model, we faced issues as we were constantly running out of ram as there were a lot of variables in our output layer. To test that out we limit our output prediction layer to the first 250 unique labels. Currently we have 3 items in our data loader-1. Imagedata (contains images), 2. Numerical labels 3. Named label (string format).

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

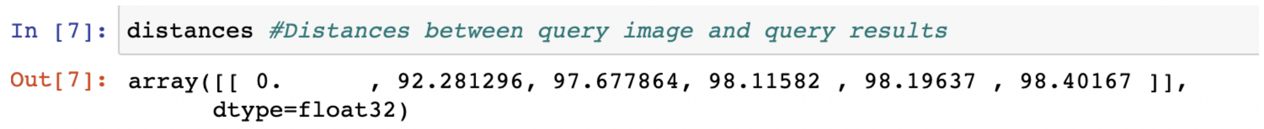
For reverse image search implementation we would not need the output layer or the classification layer. Our main aim here is only the feature extraction. To do this we remove the last 2 layers of our Baseline model, that way only features are extracted from the images. We save these flattened features.

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=6, algorithm='auto', metric='minkowski').fit(features)

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

From this we are trying to find 5 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-



```
In [7]:  distances #Distances between query image and query results

Out[7]:  array([[ 0.       , 92.281296, 97.677864, 98.11582 , 98.19637 , 98.40167 ]],
               dtype=float32)
```

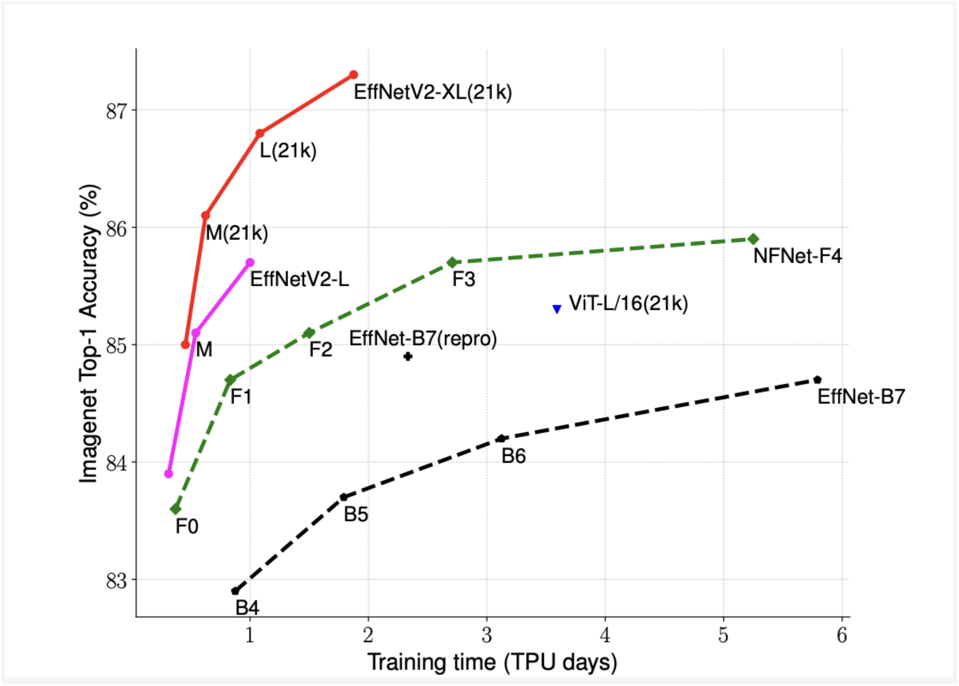
As you can see from the above output the minkowski distance metrics shows the nearest 5 images to the image we selected or queried.Our aim for the Improvement model will be to be able to reduce the distance between the queried images and nearest found images.

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

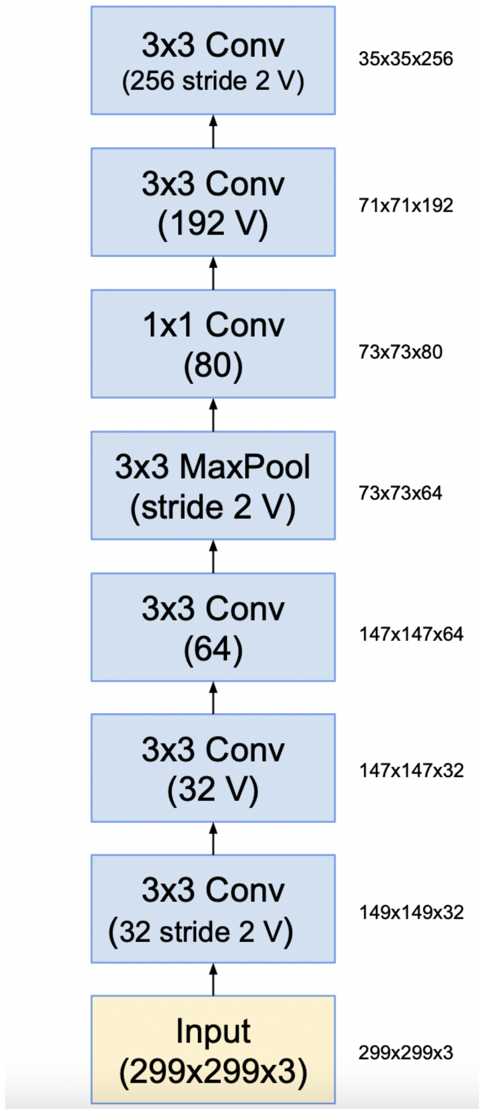
The weights= ‘imagenet’ means we are pre-training on ImageNet. The inclue_top parameter means whether we include a fully connected (FC) layer on the top of the network or not. In this case we have not included the FC layer on the top. The input_shape should have 3 input channels which in this case is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

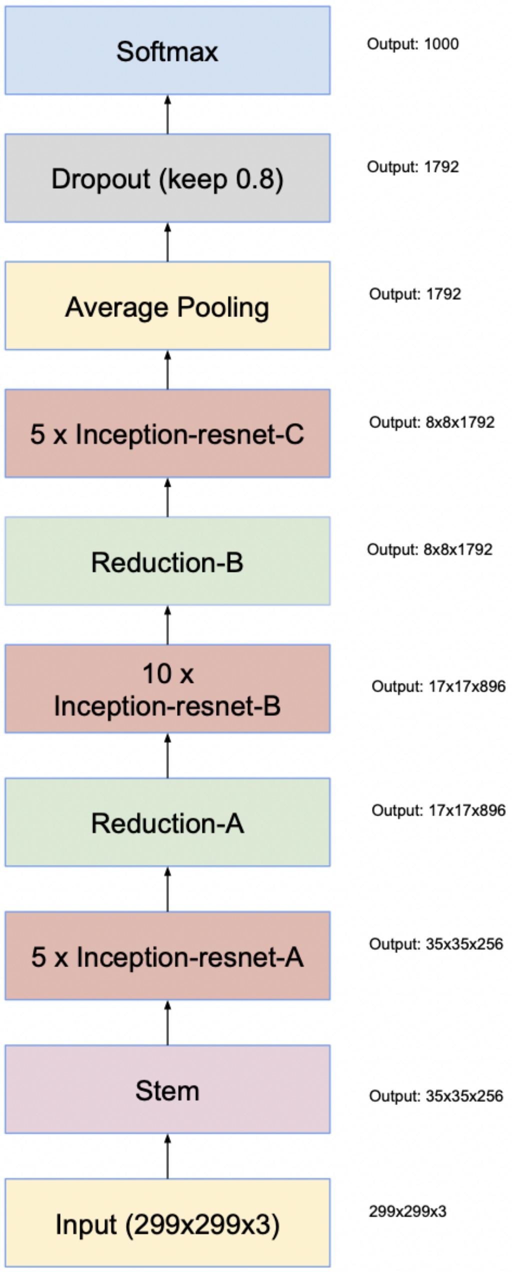
ResNet V2

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.



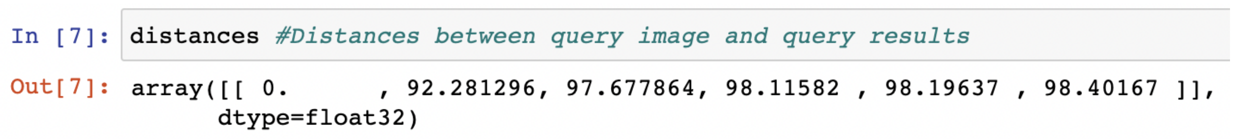


A) The stem of Inception ResNet V2 B) The schema for Inception ResNet V2

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.

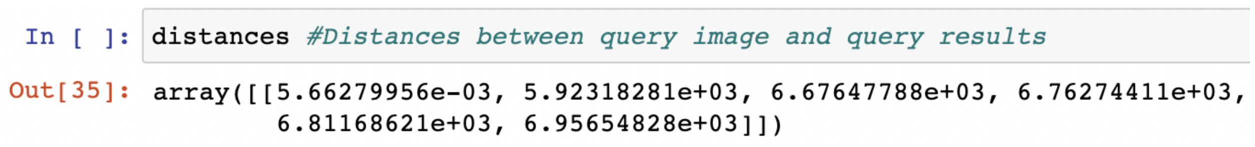
Improvement Metrics

Baseline model distance



```
In [7]: distances #Distances between query image and query results
Out[7]: array([[ 0.        , 92.281296, 97.677864, 98.11582 , 98.19637 , 98.40167 ]],
              dtype=float32)
```

Improvement Model with Resnet V2



```
In [ ]: distances #Distances between query image and query results
Out[35]: array([[5.66279956e-03, 5.92318281e+03, 6.67647788e+03, 6.76274411e+03,
                 6.81168621e+03, 6.95654828e+03]])
```

EXPERIMENTS

https://github.com/DevonARP/NYU_AI_6613

Improvements

Comparing the baseline model and improvement model

CONCLUSION

We then visualize these images in order to compare our actual results.