Face Detection From Images Using VGGNet

Anurag Marwah (am8482@nyu.edu)

(Team 1 – https://github.com/anuragmarwah/Face-Detection-Using-VGGNet)

***Abstract* – Object detection using image classification is becoming increasingly used for both general and specific purposes, with face detection being one of the primary. With some evaluation, it becomes clear that to implement a face detection classifier, a convolutional neural network (CNN) is a computationally fast and accurate choice. Pre-trained CNNs are available, which can either be used as is for general image prediction, or be further fine-tuned for a specific binary or multi-class classification. If such a CNN is fine-tuned over a dataset of faces, the resulting model can indicate the presence and location of single or multiple faces in a test image with significant accuracy.**

***Keywords* – Image Classification, Face Detection, Convolutional Neural Network, Transfer Learning**

I. INTRODUCTION

The applications of image processing in object detection are taking strides with the availability of some very powerful Python packages. Over the past few years, this is further fueled by the development and availability of pre-trained convolutional neural networks (CNNs). These CNNs were originally designed, in various years, to participate in the annual ImageNet competition [1], where millions of images are classified into one thousand categories. The pre-trained weights of these networks can be downloaded from the Keras website [2]. The proposed design uses a VGG16 Convolutional Neural Network to perform face detection.

To achieve this task, python programming is used and there are three stages of development: download the VGGNet model, train the top layer of the model over a dataset of facial images, and predict the presence of faces in parts of a test image. If the model predicts that there is a face, a box is drawn around it.

II. PROBLEM FORMULATION

DOWNLOAD MODEL

The VGG16-Net is downloaded from Keras without the top layer, with an input shape of (None, 64, 64, 3). This input size is chosen considering CPU computational abilities during model training. After downloading the model, the existing convolutional layers are frozen so that they are not trained again. Then, 3 layers are added to the model, 1 Flatten and 2 Dense layers, and at the output, sigmoid activation is used to get binary classification. Thus, the output shape of the model is (None, 1). These new layers are later trained over the training data. The total number of parameters in the model is 15,239,489 while the number of trainable parameters is 524,801.

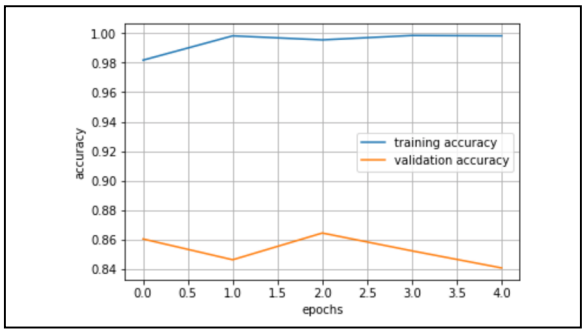
TRAIN MODEL

The dataset for facial images is built up from three sources: the FEI Face Database by Central University of FEI, Brazil [3], the Database of Faces by AT&T Laboratories Cambridge [4], and from ImageNet using the Python Flickr API [5]. The AT&T dataset is in grayscale, while the other two are in RGB color. Data generators with batch size 32 are used for the train and test datasets, each having two classes, named as ‘face’ and ‘notaface’. The ‘face’ dataset mainly

comprises of close-up images of the face of various subjects from various angles. The 'notaface' dataset is built up from images downloaded from ImageNet from five categories; namely animal, bird, building, car, and tree. For the model, binary cross-entropy loss is considered, along with accuracy as the metric, and Adam optimizer is used with a learning rate of 0.001. The model is trained for 5 epochs. The training and validation accuracy is observed and plotted. The model training completes with a final validation accuracy of 0.8407.

PREDICT FACES

The trained model is evaluated for test images to detect and indicate the presence of faces. A sliding window is used



to get part of the image. The size of the sliding window is (64, 64, 3). This size is chosen to match with the input size of the trained neural network. The image data in each window is passed as input to the neural network and the model performs binary classification at a threshold of 0.5 between 'face' and 'notaface'. The window is moved Left to Right, and Top to Bottom, with a step-size of 8. This provides a smooth transition without too much overlapping of adjacent windows. To counter the effect of variable face sizes in an image, a readily available scikit-image function of pyramid_gaussian () is used [6]. This function sub-samples the image by the given scale factor. This way, the same window size, and thus the same neural network model, can be used to detect faces of different sizes in the same image.

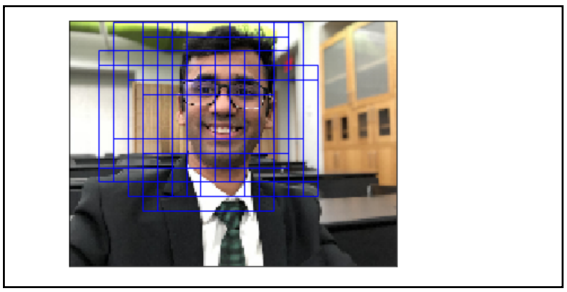
The pseudo-code of the algorithm that is used to predict faces is given below:

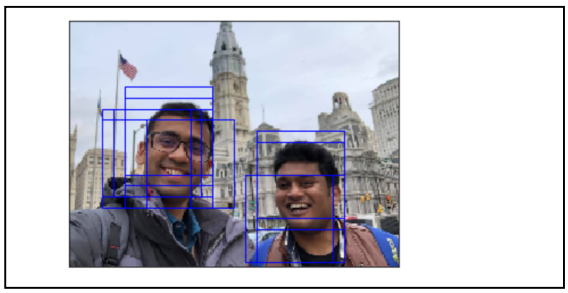
ALGORITHM 1: PSEUDO-CODE FOR FACE PREDICTION

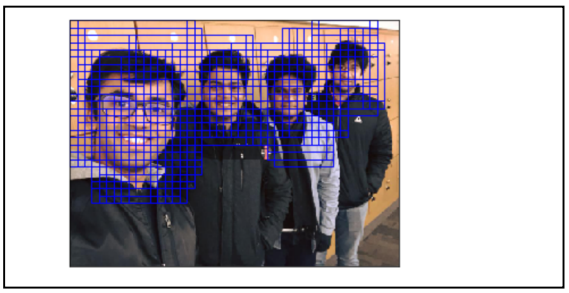
```
1  Import Model as M
2  Build Image_Pyramid
3  for Image in Image_Pyramid do:
4     for Window in Image do:
5        // Predict Face in Window
         Prediction = M.Predict(Window)
6        if Prediction == TRUE then:
7           Draw a Box around Window
8        end if
9     end for
10 end for
```

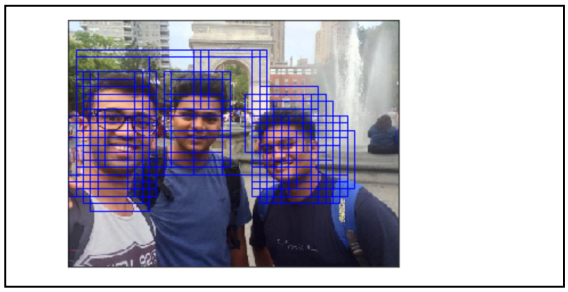
III. EVALUATION AND RESULTS

The trained model is used to predict faces in ten different test images. The test images are chosen such that it covers a range of single and multiple faces per image. A scale factor of 2.0 is used for the image pyramid. Most of the test images are in the aspect ratio of 4:3. As mentioned before, the window size is (64, 64, 3) and the step size for the window is 8. For most of the images and for most of the faces in the images, the model predicts the faces very well. Some of the results are attached below. The rest of the results are available on the Github Link of this project [7].









IV. Conclusion and Future Scope

Using a pre-trained VGGNet, and by further fine-tuning it, I was able to perform face detection with high accuracy. A useful extension of this implementation can be the inclusion of non-maximum suppression, so that there exists just one box for each face in the output.

Acknowledgment

I am very thankful to Professor Yao Wang for the close guidance on the design and review of the model.

References

1. ImageNet Competition - http://www.image-net.org/challenges/LSVRC/
2. Keras Applications - https://keras.io/applications/
3. FEI Face Database - http://fei.edu.br/~cet/facedatabase.html
4. AT&T: The Database of Faces - http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
5. Python Flickr API - https://stuvel.eu/flickrapi-doc/
6. Scikit-Image Pyramid_Gaussian website - http://scikit-image.org/docs/dev/api/skimage.transform.html#pyramid-gaussian
7. Github Link of Project - https://github.com/anuragmarwah/Face-Detection-Using-VGGNet