Face Detection with Deep Learning

1. INTRODUCTION

Face detection is the technique to identify human faces in digital photos/videos. This uses Artificial Intelligence to do so. Face detection uses machine learning and deep learning algorithms, statistical analysis, image processing techniques to pinpoint human faces within larger images. These larger images may include objects such as landscapes, sceneries, and anything other than human faces.

Some of the applications of face detection can be in 24x7 surveillance, biometrics, law enforcement, social media, and many more. The methods for face detection are that have been used and developed over time are:

1. Viola-Jones Algorithm
2. Knowledge/rule-based face detection
3. Feature Based or feature invariant
4. Template matching
5. Appearance based

The state-of-the-art methods for face detection involve the use of Deep learning techniques such as Multi-Task Cascade Convolutional Neural Network (MTCNN), and Single shot detectors like YOLO. Extended applications include predicting the face position using Kalman Filter tracking. This increases the face detection rate and meet the real time detection requirements.

With the technological boom Deep Learning has paved the way for creating accurate models that can detect faces with a very high precision accuracy. Some of these methods that proved to perform better than “traditional” methods mentioned above are:

a. Training simple classifier [3]
b. Convolutional Neural Networks combined with Kalman Filtering (Ren et al. (2017)): CNNs are used here to detect the face in a video. But when a face is largely deflected or severely occluded, Kalman Filter tracking is used to predict face position. This increases the face detection rate and meet the real time detection requirements.
c. Deep Cascaded detection method – exploits bounding-box regression, a localization technique, to approach the detection of potential faces in images. This involves cascaded architecture with 3 stages of deep convolutional networks to predict existence of faces.

Even though Face detection has been prominent in the industry for quite some time, it still faces a few challenges that the researchers and engineers look to improve. Listed below are the reasons that mainly affect face detection and reduce the accuracy and face detection rate. [1]:

a. Odd human expressions in a digital image
b. Face occlusion: face hidden by other objects
c. Lighting effects – varying Illuminations
d. Complex backgrounds
e. Too many faces in the image
f. Low resolution images
g. Varying skin color
h. Face orientation
i. Distance from camera

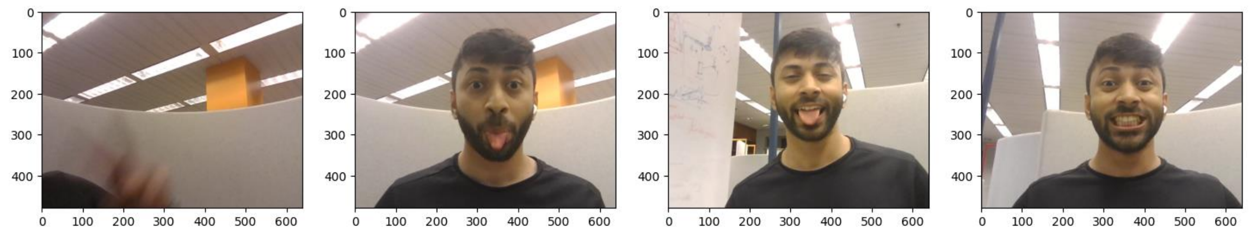
Latest advances in technology have propelled people and industries to offer the face detection as a service in the form of APIs. Some of the notable ones are Amazon Recognition, Microsoft Face API, IBM Watson visual recognition, Google cloud vision, etc. The other revolution for such applications was led by the introduction of Deep Learning which has taken the world of computer vision by storm. Some of the latest and most accurate and precise face detection models are:

a. DeepFace
b. ArcFace
c. YOLO
d. FaceNet
e. Dlib
f. MTCNN

These advanced methods have been utilized in developing the state-of-the-art face detection models like the single shot detectors and two-shot detectors. In this project we develop a two-shot learning method to detect faces.

2. DATASET/DATABASES FOR FACE DETECTION

For the purpose of developing the face detection model, a custom dataset comprising of photos taken via laptop's webcam has been prepared, and would be used to train a deep learning model for the downstream task of face detection. Sample images of our custom dataset is shown below:



The dataset contains images that have a face and those which do not have a face in them. Initially a total of 90 images were collected. These images were then put through the augmentation process and a final dataset was scaled to ~4000 images in total.

Other than the above method to train a face detection model some of the standard datasets/databases of different types of faces that have been used as benchmarks for the state-of-art models [1]:

[Face Recognition Homepage - Databases (face-rec.org)](face-rec.org)

Database	Website	Description
MIT dataset	http://cbcl.mit.edu/softwaredatasets/FaceData2.html	19 × 19 Gray-scale PGM format images. Training set: 2429 faces, 4548 non-faces. Test set: 472 faces, 23,573 non-faces
PIE database, CMU	www.ri.cmu.edu	A database of 41,368 images of 68 people, each person under 13 different poses, 43 different illumination conditions, and with 4 different expressions
FERET database	www.itl.nist.gov/iad/humanid/feret/feret_master.html	It consists of 14,051 eight-bit gray-scale images of human heads with views ranging from frontal to left and right profiles
The Yale face database	www.face-rec.org/databases/	Contains 165 gray-scale images in GIF format of 15 individuals. There are 11 images per subject, one per different facial expression or configuration: center-light, w/glasses, happy, left-light, w/no

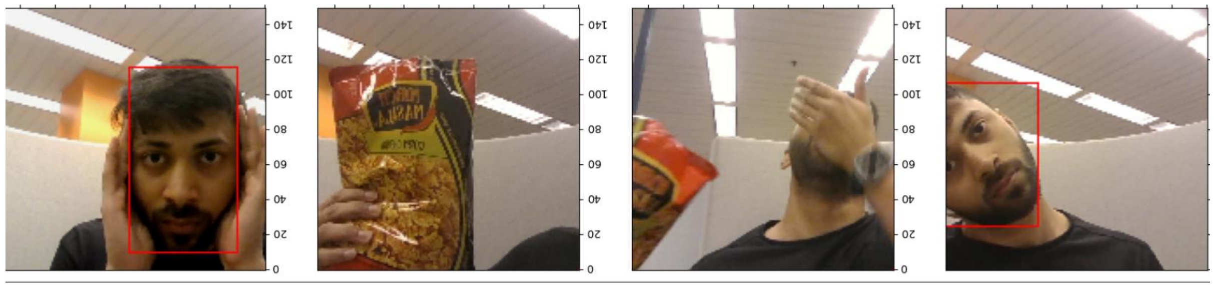
Database	Website	Description
		glasses, normal, right-light, sad, sleepy, surprised, and wink
Indian face database	www.pics.stir.ac.uk/Other_face_databases.htm	11 images of each of 39 men, 22 women from Indian Institute of Technology Kanpur
AR database	http://www2.ece.ohio-state.edu/~aleix/	It contains over 4000 color images corresponding to 126 people's faces (70 men and 56 women). Features based on frontal view faces with different facial expressions, illumination conditions, and occlusions (sun glasses and scarf)
SCface—surveillance cameras face database	www.scface.org	Images were taken in uncontrolled indoor environment using five video surveillance cameras of various qualities. Database contains 4160 static images (in visible and infrared spectrum) of 130 subjects

3. DATA DESCRIPTION

A custom dataset containing live images taken through the webcam at different poses and distances is utilized. The process of data collection is described below:

3.1. 3 batches of 30 images are captured through the web camera of the laptop/personal device. This is done using the Python “OpenCV” library.

3.2. Using python “labelme” software program, the captured images were labeled as having a “face” or not having a face. Iteratively bounding boxes were drawn for each image containing a face and a json file of the labeled images was created. An example of positive and negative labeled image sample is shown below:



The samples that contained the face were labeled as shown above and the ones that did not contain a face were not.

3.3. Each file was assigned a unique id using the uuid python module. Here are a few samples of the image identifiers assigned to each sample in our data:

- 2de37d51-fa4b-414f-acd8-3cf70b5cb75d.jpg
- 6c0ad2d9-810b-400e-98a2-a5dd6f03d114.jpg

3.4. Of the 90 images collected, 68 images had faces in them and the rest did not. These images were split into train, validation, and test images in the ratio of 70-15-15 percent. This resulted in 62 images in training dataset, 15 in test dataset, and 13 in the validation dataset.

3.5. Image augmentation was carried out using the albumentations library and the final dataset after augmentation was scaled to 3720 images. These images were then used to train and validate the face detection model.

4. PRE-PROCESSING

After the initially collected images are split into train, test, and validation sets data augmentation is carried out. This is particularly done so that the model can be exposed to variations of images while training. This enables the model to become more generalized and adaptive to a wide range of scenario's during deployment.

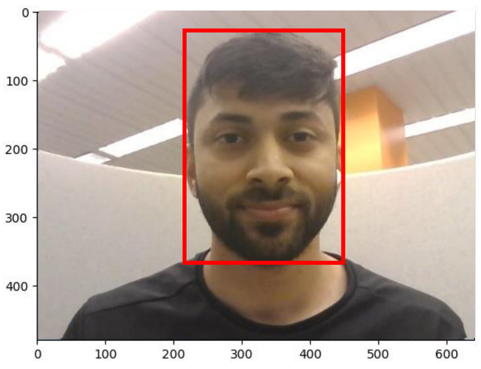
There are mainly 2 types of data augmentation techniques: geometric and photometric. Geometric transformations involve changing the positions of image pixels. This includes reflection, rotations, translation, flipping, etc. Photometric transformations involve changing the RGB channels i.e. by shifting the pixel colors to new values. This includes approaches like Gray scaling, color jittering, filtering, lighting perturbation, noise adding, vignetting, contrast adjustment, etc.

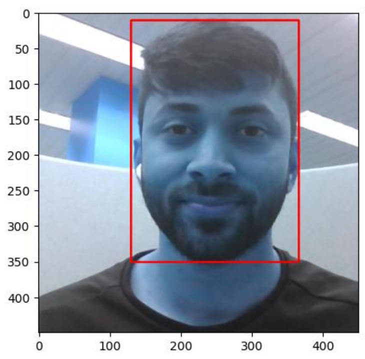
For our model, there were six types of image augmentations applied. These augmentations were applied at random and combined to obtain complex image transformations:

i. Random Cropping: crops the image at random maintaining a fixed image size (in our case 150 x 150)
ii. Horizontal Flipping: This inverts the image horizontally.
iii. Brightness Contrast: With this the contrast value of the image is controlled.

iv. Random Gamma variation: This property controls the overall brightness of an image.
v. RGB Shift: This is a photometric augmentation where pixel colors were changed.
vi. Vertical Flipping: This inverts the images vertically.

These transformations were applied to all the images in the train, test, and validation folder. The resulting size of the image dataset was 3720 images in the train set, 780 in the test set, and 900 in the validation set. A positive sample (image containing a face) of augmented image where horizontal flip, random gamma value, brightness contrast, and random cropping have been applied to a single image can be seen below:





5. MODELS IMPLEMENTED

The model used to train our face detection algorithm is a VGG16 model on which 2 more layers for classifying and detecting the face in the image are added. On the core, the basic

5.1. VGG16 Architecture Overview

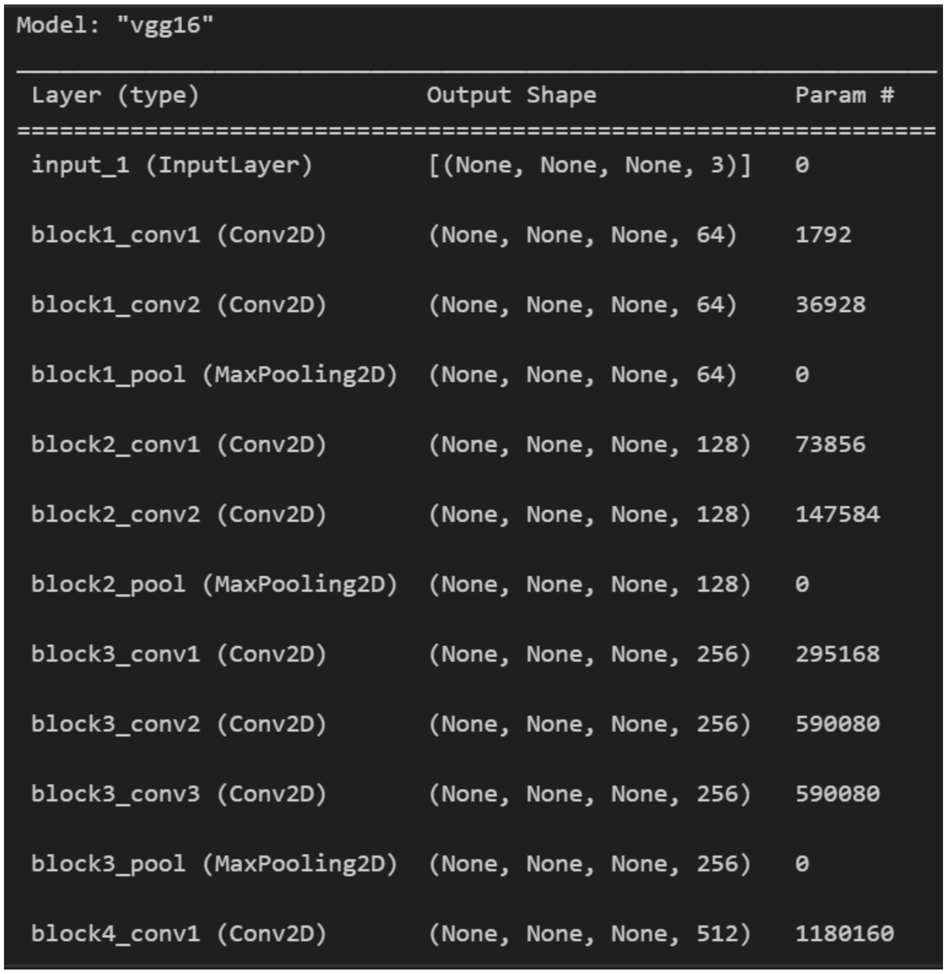
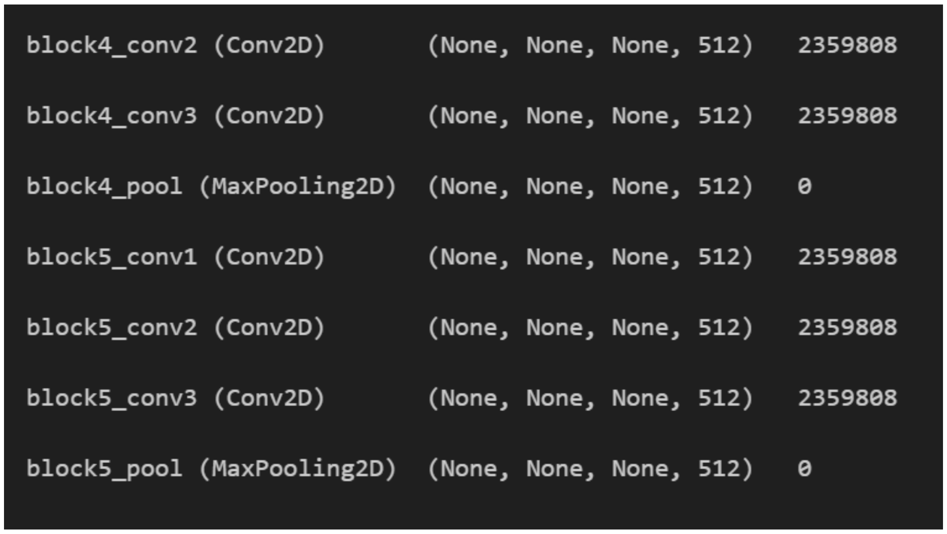
VGG16 model that has been trained on ImageNet dataset. We customize the pretrained VGG16 model for face detection. We first explore the VGG16 architecture to understand how it will be able to implement face detection as a downstream application.

The VGG16 model consists of a total of 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional layers are arranged in sequential blocks, and each block has multiple convolutional layers followed by a max-pooling layer. The fully connected layers at the end are responsible for the final classification. With the original VGG16 architecture, the input size of the image is usually 224 x 224 pixels but this can be c=varied depending on the application through the TensorFlow keras module.

The convolutional layers in VGG16 use small 3x3 filters with a stride of 1 and a padding of 1 to preserve the spatial dimensions. These layers learn hierarchical representations of the input images, capturing increasingly complex features as the network deepens.

After each set of convolutional layers, VGG16 includes a max-pooling layer with a 2x2 filter and a stride of 2. Max-pooling reduces the spatial dimensions of the feature maps while retaining the most salient features.

VGG16 has three fully connected layers with 4096 units each. These layers serve as the classifier, transforming the high-level features learned by the convolutional layers into class probabilities. Below is the model summary obtained after the model was loaded in the python environment.

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160
```

```
 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)  (None, None, None, 512)   0
```

5.2. Parameters & Activation Functions

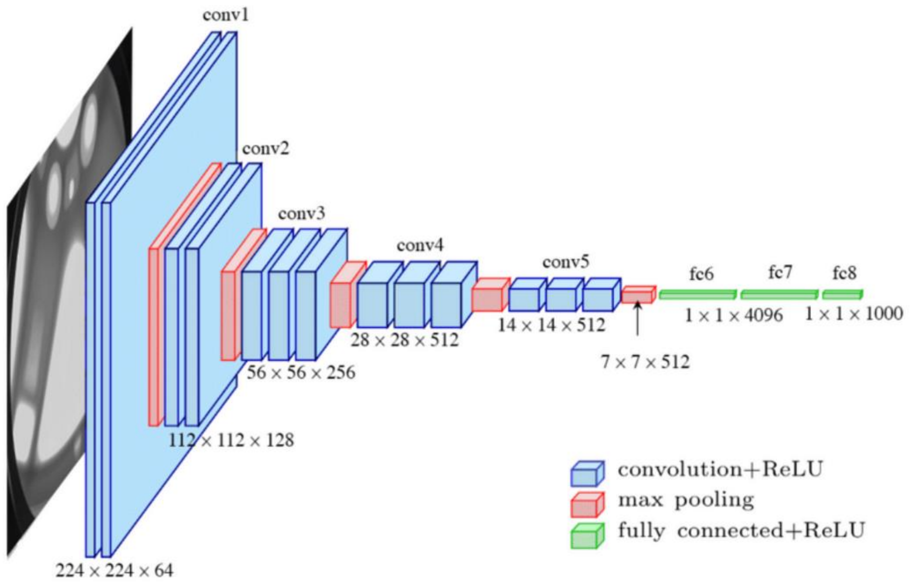
VGG16 has many parameters, approximately 138 million. The majority of the parameters are concentrated in the fully connected layers, making VGG16 a relatively deeper and more parameter-intensive model compared to its predecessors.

The rectified linear unit (ReLU) activation function is used after each convolutional and fully connected layer, introducing non-linearity to the network.

The final layer of VGG16 uses a softmax activation function to produce class probabilities. It generates a probability distribution over the classes, indicating the likelihood of the input image belonging to each class.

VGG16 has been widely used as a backbone architecture in various computer vision tasks, including image classification, object detection, and image segmentation. Although it has been surpassed in terms of performance by more recent and complex architectures, VGG16 remains a popular choice due to its simplicity, ease of implementation, and solid performance on a wide range of visual recognition tasks.

A preview of how a VGG16 model looks like can be seen below.



6. METHODOLOGY

With pre-processing out of the way. The next steps constitute the preparation of the dataset for training and optimizing the model. An overview of the next steps incorporated to develop the face detection model is mentioned below:

6.1. Converting images and labels to TensorFlow format

Since, we will be using TensorFlow and keras to train our model, we begin by converting our data (images and labels) to a TensorFlow dataset. A TensorFlow dataset represents a sequence of elements where each element here comprises of batch of one or more components. To make it simpler, an example for an image pipeline, an element of the TF Dataset can be a single training sample, with a pair of tensor components representing the image and its label. For the purpose of creating a dataset comprising of images and labels for face detection using data collected in real-time, a data transformation constructs the dataset from one or more tf.data.Dataset objects. The images and labels are converted to a TensorFlow dataset individually and in an element wise manner.

To convert the images to a Tf dataset, we use the Dataset.map transformation that apply a function to each element. Each image is first resized to 150 x 150 pixels followed by standardizing the values i.e., by dividing them by 255. This results in pixels values being in range from 0 to 1. Each of train, test and validation dataset are created separately.

After the images, the labels are converted to Tf Dataset using data transformations as well. First a label loading function is defined which extracts the class of the corresponding image and the coordinates of the bounding box where the face is present. This is followed by a transformation which converts the extracted values to a list of the given datatype.

6.2. Combining TensorFlow images and labels into a dataset for training and validation

With the Tf Datasets of images and labels in place, both are zipped together to form a complete dataset which will be used to train the face detection model. The elements of the dataset are shuffled at random to eliminate the possibility of overfitting due to the model memorizing the order of the elements. This also helps the model to learn the general pattern existing in the dataset. The elements are then batched in groups of 8 (batch size). This helps in the training being less computationally expensive and results in shortening the training times as multiple images would be processed at once and the network would be updated accordingly.

6.3. Building the Deep Neural Network for training (with pre-trained VGG16)

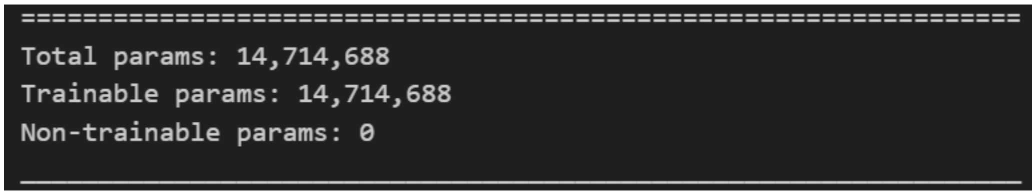
With the dataset ready to be used for training, initializing the model is a necessary step. As mentioned before, we will be using the VGG16 as our base architecture. The model head was replaced with a Dense classification layer and Dense regression layer. The Classification layer would classify the images on whether it has a face or not. The regression layer would then apply the object localization to predict the location of the face by predicting the bounding box coordinates for the face. This process is often termed as Bounding Box Regression.

Initially, a VGG16 pretrained image classification model using the Keras API has been used to classify the images based on the presence of a face or not. This is then followed by a regression model that determines the coordinates of the bounding box for the detected face. Additional layers for classification and regression are added as the final 2 layers for the task of face detection. This is a classic example of Object localization algorithm which basically identifies the object (in this case – the face) followed by the location of the object in the image by visualizing a bounding box around it.

The results of implementing the model will 5 values:

a. First set of value will be whether a face was detected or not
b. The next set will be 4 values of the X, and Y coordinates of our bounding box.

For the above to be implemented, the VGG16 model without its model head is imported from the TensorFlow keras application module. This enables us to modify the final layers of the architecture as per the downstream task. The classification layer uses the Binary cross entropy as its loss function and the sigmoid function as the activation function. Similarly, the regression layer uses the localization loss and a sigmoid activation function at the output. The input of this model is a tensor with dimensions of 150 x 150 x 3 as we resize the images to that size in the above steps. The trainable parameters can be seen in the summary below.



```
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

_________________________________________________________________
```

6.4. Define loss function, optimizer, and model hyperparameters

To train the model, the hyperparameters that were experimented with are batches per epoch, learning rate decay. The batches per epoch was set to be the same as the length of the training set and a decay rate was used because it is desired for the neural network to slow learning until a local minimum is reached. This helps in the optimizing the model and in generalizing the model.

Apart from this the Adam optimizer is utilized with a learning rate of 0.0001. The Adam optimizer is used because of its straightforward implementation, and computational efficiency. Additionally, the optimizer uses very low memory compared to others and it is invariant to diagonal rescaling of gradients while optimizing the model.

The Loss functions used for this use case is:

a. Binary Cross Entropy for classification

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log(1 - \hat{Y}_i))$$

b. Localization Loss function: this measures the error of the X and Y coordinates and the height and width of the bounding box with the actual and predicted values. The function simply calculates the Euclidean distance between the points and is represented as

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

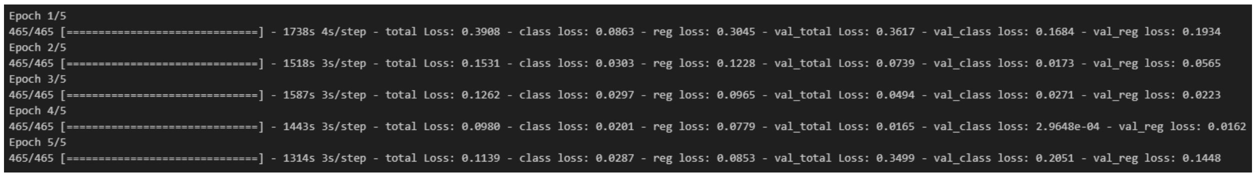
where

$1_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

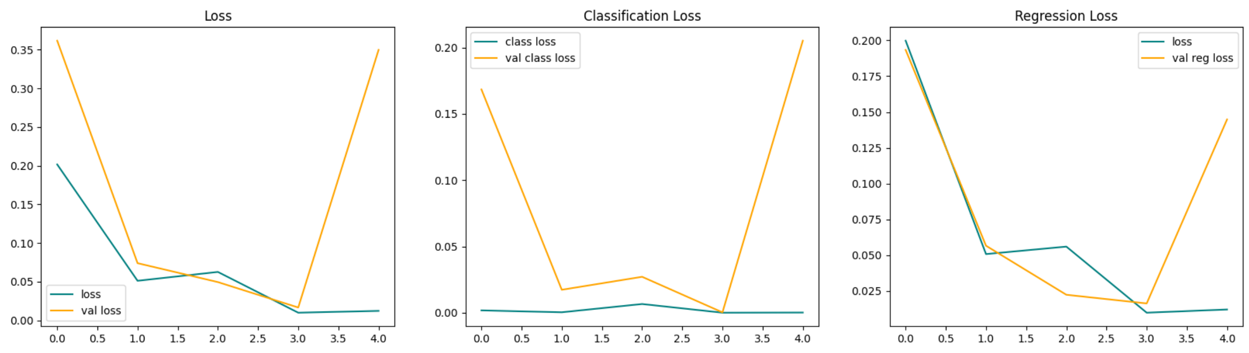
6.5. *Training the Model*

With the parameters and hyperparameters being defined, the model is trained for 5 epochs due to the computational resources. The training was carried out on the *Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz Processor.* To track the model instances being trained a log directory is also created while training using the tensorboard callback function in the fit function.

Post training the training and validation performance can be plotted. It can be seen from the graphs that the validation spikes at the 5th epoch. The major reason can be attributed to limited training of the model. The further explanation to this can be overfitting and the lack of variety in the data present. But this would currently be inconclusive as the model was not trained for enough epochs and did not reach the tolerance limit i.e., the model was still learning.

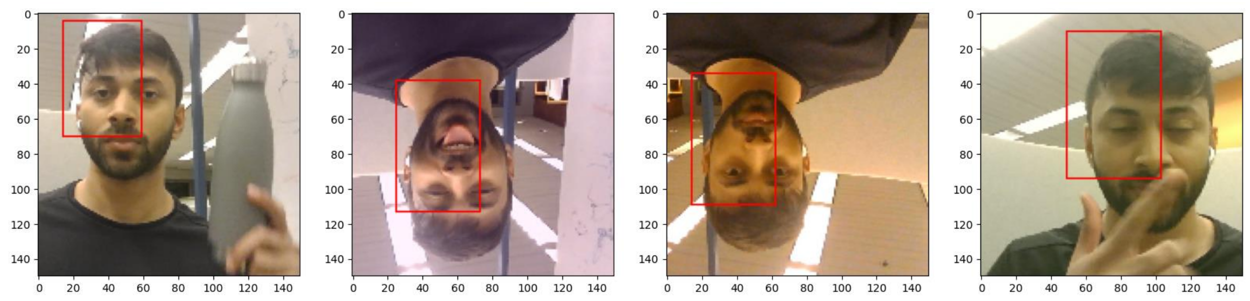


```
Epoch 1/5
465/465 [==============================] - 1738s 4s/step - total Loss: 0.3908 - class loss: 0.0863 - reg loss: 0.3045 - val_total Loss: 0.3617 - val_class loss: 0.1684 - val_reg loss: 0.1934
Epoch 2/5
465/465 [==============================] - 1518s 3s/step - total Loss: 0.1531 - class loss: 0.0303 - reg loss: 0.1228 - val_total Loss: 0.0739 - val_class loss: 0.0173 - val_reg loss: 0.0565
Epoch 3/5
465/465 [==============================] - 1587s 3s/step - total Loss: 0.1262 - class loss: 0.0297 - reg loss: 0.0965 - val_total Loss: 0.0494 - val_class loss: 0.0271 - val_reg loss: 0.0223
Epoch 4/5
465/465 [==============================] - 1443s 3s/step - total Loss: 0.0980 - class loss: 0.0201 - reg loss: 0.0779 - val_total Loss: 0.0165 - val_class loss: 2.9648e-04 - val_reg loss: 0.0162
Epoch 5/5
465/465 [==============================] - 1314s 3s/step - total Loss: 0.1139 - class loss: 0.0287 - reg loss: 0.0853 - val_total Loss: 0.3499 - val_class loss: 0.2051 - val_reg loss: 0.1448
```



6.6. *Test the model*

The test performance is first observed by face detection on the test sets. The results obtain by be visually inspected and the performance is seen to be low. This can be attributed to the limited model training as it was trained for only 5 epochs. Another reason can be the limited size of dataset when compared to the large deep neural networks and the state of art models like MTCNN, DNN, YOLO. A glimpse of the testing performance can be noted in the image below where it is seen that the predicted bounding boxes are shifted to either side of the face.



6.7. *Implement face detection in real-time using OpenCV*

With the partially trained model, it was implemented for real-time face detection using the opencv module. The threshold value set for this purpose is 0.5. This means that the bounding box would be displayed if the predicted probability of a face being present in the captured image instance from the real-time feed is greater than 0.5. This threshold value is flexible and can be varied depending on where face detection is applied in real-life. A small sample of the real-time face detection can be seen in the video uploaded “Real-time Video Face Detection.”

7. CONCLUSIONS & LIMITATIONS

By using VGG16 as the base architecture for the face detection task, it is expected to perform with high accuracy and to a certain extent and limited performance the model did perform well when executed in real time using OpenCV. By implementing this model, an important skill of transfer was shown where any pretrained model can be modified for any downstream task and result in good performance. Also, VGG16 deep architecture allows it to identify complex feature representations which can be advantageous for face detection.

The current face detection model was only trained on images of a single person, with limited image transformations, and with limited computational resources. The model was also trained on 5 epochs only considering the computational limitations. Due to this the model could not be immediately deployed to a software application. Another aspect can be the limitations of VGG16 with face detection as face detection requires localized and spatially precise predictions which might be the strength of a VGG16 model. The fixed-size input images and max-pooling layers in VGG16 may cause some loss of spatial resolution and potentially affect the accuracy of detecting small or partially occluded faces.

8. FUTURE WORK

The subsequent work would be to enhance the face detection model by training it on images that contain 1 or more different faces rather than the face of a single person. Moreover, increased transformations/augmentations like color jittering, filtering, noise addition, etc. can be applied to the static images for training purposes. Following the basic augmentations custom augmentations that transform only particular components of the image can be applied. One such example is changing the hairstyle, makeup, and attributes of the person face(s) in the image. This can be achieved by the use GANs. These pre-processing steps can further enhance the model generalization and to a certain extent the performance as well.

9. REFERENCES

[1]. Kumar, A., Kaur, A. & Kumar, M. Face detection techniques: a review. Artif Intell Rev 52, 927–948 (2019). https://doi.org/10.1007/s10462-018-9650-2

[2]. P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517

[3]. Mukherjee S, Saha S, Lahiri S, Das A, Bhunia AK, Konwer A, Chakraborty A (2017) Convolutional neural network based Face detection. In: Proceeding of 1st international conference on electronics, materials engineering and nano-technology, pp 1–5.

[4]. Ren Z, Yang S, Zou F, Yang F, Luan C, Li K (2017) A face tracking framework based on convolutional neural networks and Kalman filter. In: Proceeding of 8th IEEE international conference on software engineering and service science, pp 410–413

[5]. www.tensorflow.org