CAP5415: Course Project
Real Time Face Detection in Videos

Aparna Mahendrakumar Burhade (5499423)

Barkha Amarlal Chainani (5499364)

Abstract:

In this project, we apply deep learning concepts to detect faces in real time. For this purpose, we built our own dataset by collecting images using opencv and our webcam and labeling them using a library called labelme. We then augmented our data by using a library function called albumentation which basically allows us to have 30 times more data than we already had in order to train our model. Before we trained our model it was essential to calculate the losses to compare the losses before and after training. We then built and trained our model using the Keras functional API and then after successfully training our model we tested this out in real time where we were able to detect our faces.

1. Introduction:

Computer vision is a field of Artificial Intelligence that enables computers and systems to extract useful information from digital photos, videos, and other visual inputs and to conduct actions or offer recommendations in response to that information. If AI gives computers the ability to think, computer vision gives them the ability to see, observe, and comprehend. Face detection is a method for computers to detect and recognize human faces that is based on computer vision. When combined with biometric security systems, this technology allows for real-time surveillance and tracking of people. It now plays an important role as the first step in many key applications -- including face tracking, face analysis and facial recognition. Face detection has a significant effect on how sequential operations will perform in the application. Detecting faces in pictures can be complicated due to the variability of factors such as pose, expression, position and orientation, skin color and pixel values, the presence of glasses or facial hair, and differences in camera gain, lighting conditions and image resolution. Recent years have brought advances in face detection using deep learning, which presents the advantage of significantly outperforming traditional computer vision methods.

2.Method:

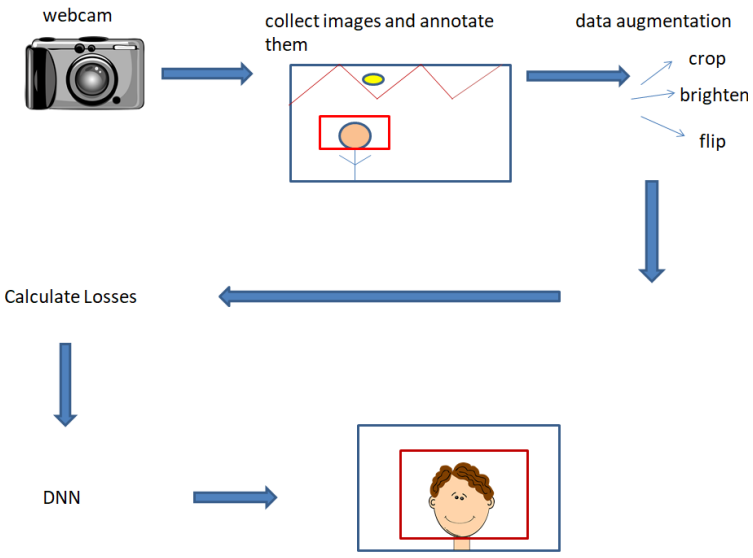


Figure a.

The overall model architecture is illustrated in the figure (figure a) shown above.

1. We first start by building our own data set by collecting images using a webcam.
2. The next thing we do is annotating the images.Annotation is the technique of drawing a bounding box around the face and in order to do that we use a library called labelme.
3. After annotating the images, we use a technique called data augmentation where we use a library called albumentation which basically allows us to take our dataset and apply random cropping, random brightness, random flips etc. This allows us to have many time more the data than we already have which is enough to train our model.
4. Next what we do is calculating the losses. For the classification component we calculate a loss called the Binarycrossentropy loss and the second loss is the localization loss. This loss basically ensures how close the box is to our face.
5. After doing this we use the Keras functional API to build and train our deep learning model.We used a VGG16 model which is a classification model for images. We use it inside of our model and add our final two layers which are our classification model and regression model to be able to give us our bounding boxes to detect our faces.
6. We then test this out in real time where we are able to detect our faces.

3. Experiments

3.1 Dataset:

The dataset used is a custom dataset that we created for this particular project. This has been done keeping in mind that the aim of the experiment was to detect faces in real time videos. Thus it was important to create and test a model that could work on datasets from real time video capture devices / webcams. This provides the project a very raw dataset that needs pre processing. This is important keeping in mind applications point of view , that most of the times the applications we perform using local cameras would require a fair amount of cleaning or preprocessing on the input stream of images/ videos.

The images were taken in different angles to allow the dataset to have a complete idea of a face. Some images did not include any faces because the dataset should be complete i.e. able to identify if a face is absent in the current frame/image.

The dataset was divided into three parts:

1. Training data
2. Testing data
3. Validation data

All the images were annotated using a library called labelme. The class used as a label was called ‘face’. To ensure completeness of the dataset , images that were blurred due to movement were also included and labeled.

Augmentation was applied by using random filters to the images , and saved in Aug_Data Folder. All three parts of the dataset were augmented. The random filters included turning the image 180 degrees or upside- down , random brightness , random crop filters. This was done to ensure that the quality of the images was random and so were the brightness and transform parameters. Albumentation library was used to perform these augmentations.

3.2 Evaluation Metrics

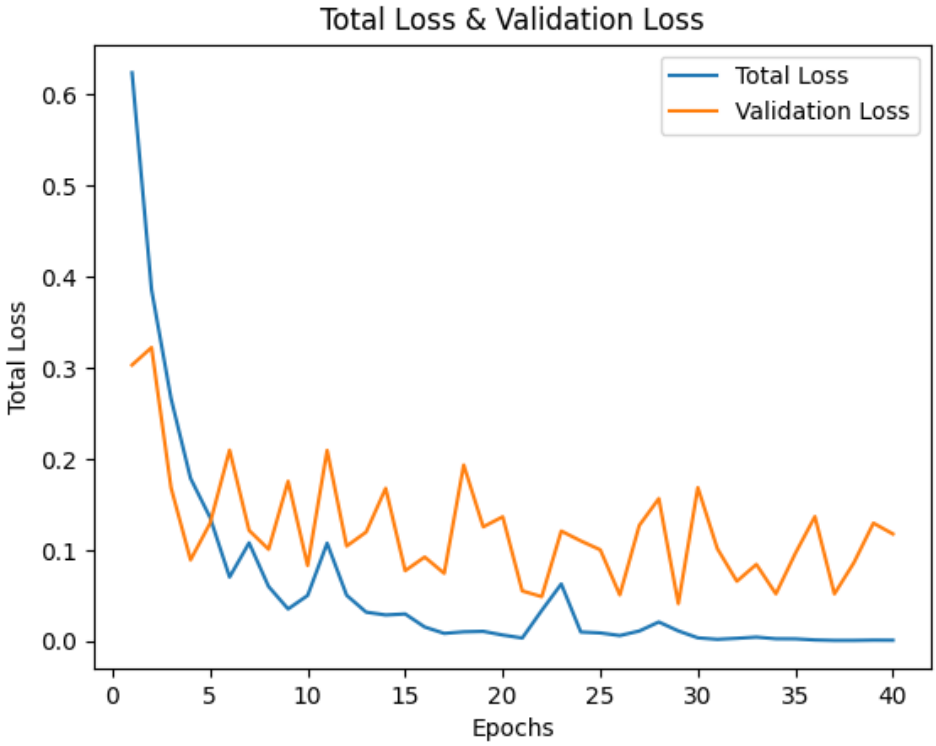
We calculate Classification and Regression losses.
Regression Loss or Localization loss is the squared difference between the true coordinates of the face and the predicted coordinates.

The classification loss is calculated by using BinaryCrossEntropy from the keras library. This library takes the log of probabilities of classifying a certain class. The probability is the chance that the object is a ‘face’. To associate loss with these probabilities a negative log is taken.

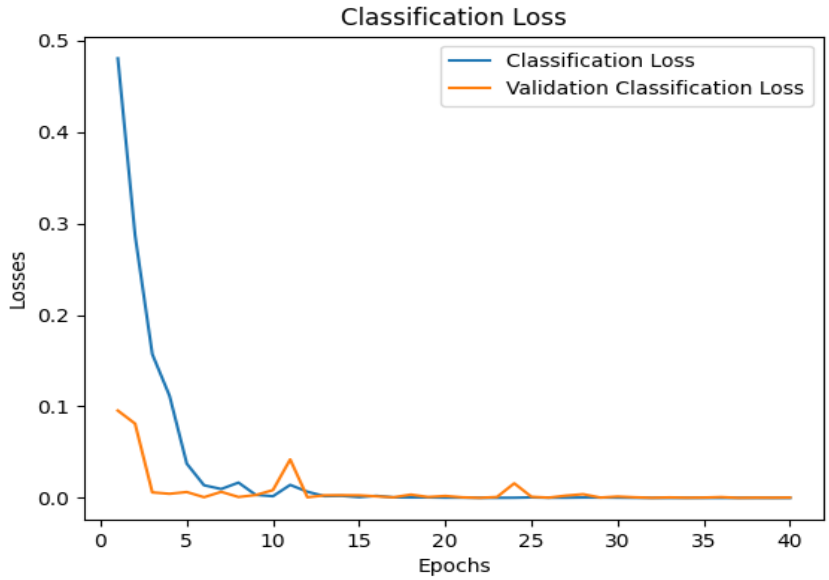
As is shown in the results below we have managed to reduce both these losses significantly after training vs before training the model.

3.3 Results

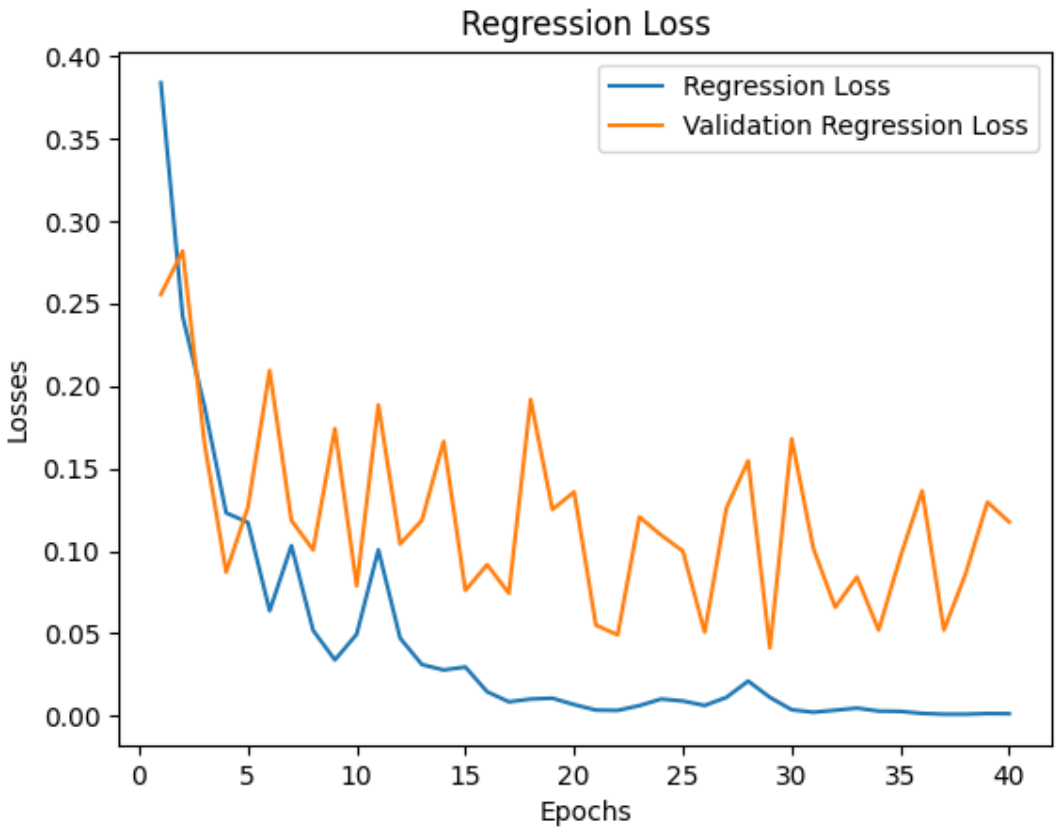
1. Total Loss



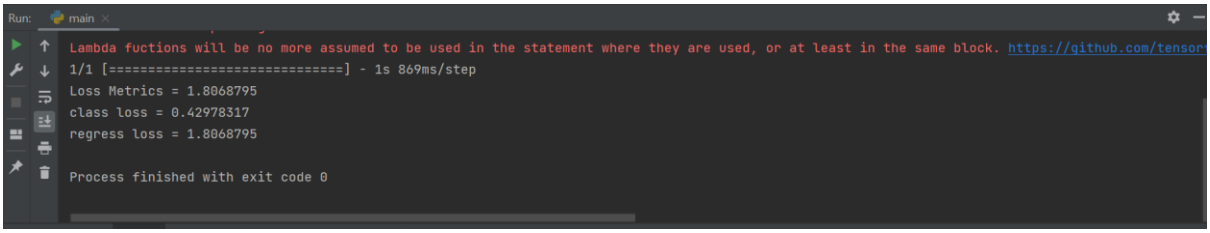
2. Classification Loss



3. Regression Loss

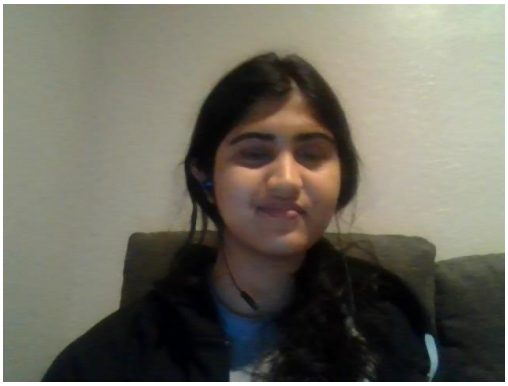
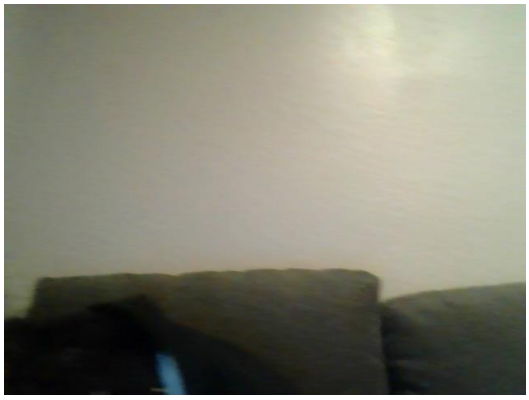
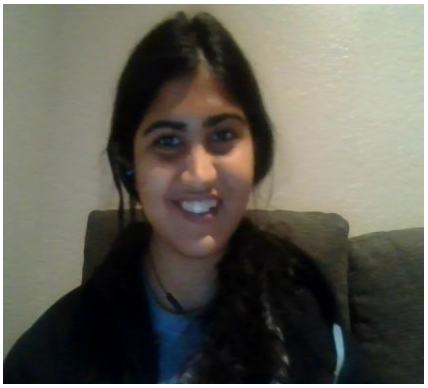
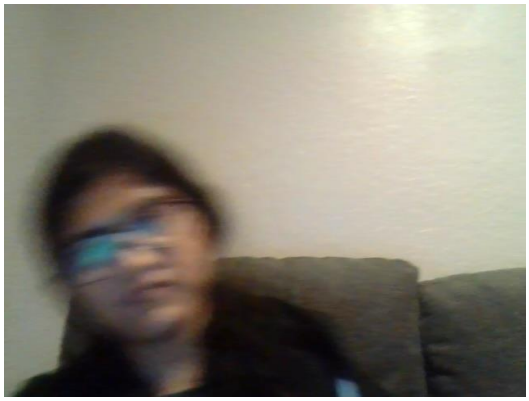


4. Loss Metrics Before Training

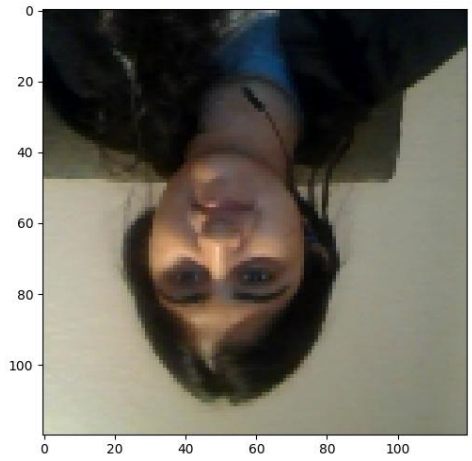


Stages wise Results

1. Sample Dataset images

2. Augmented Dataset Images



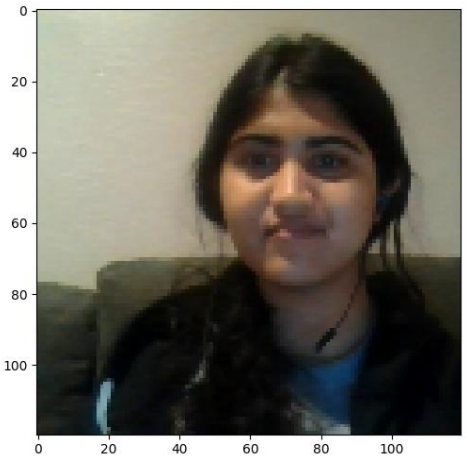


figure: Image Flip

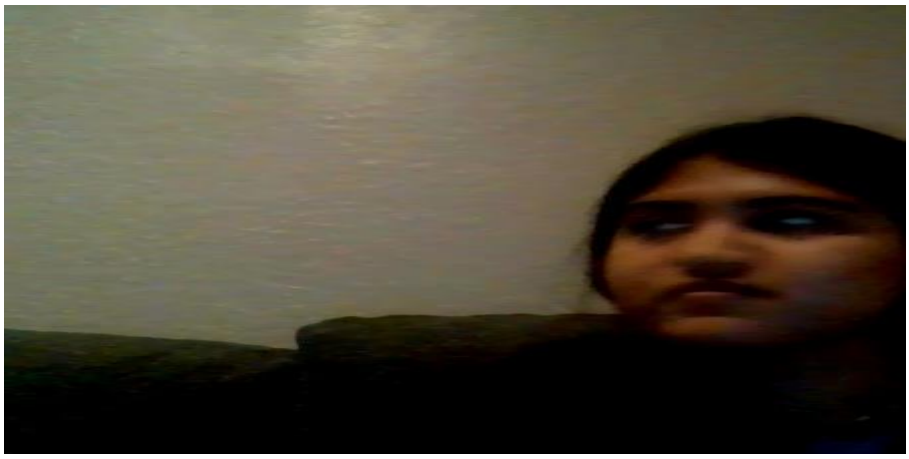
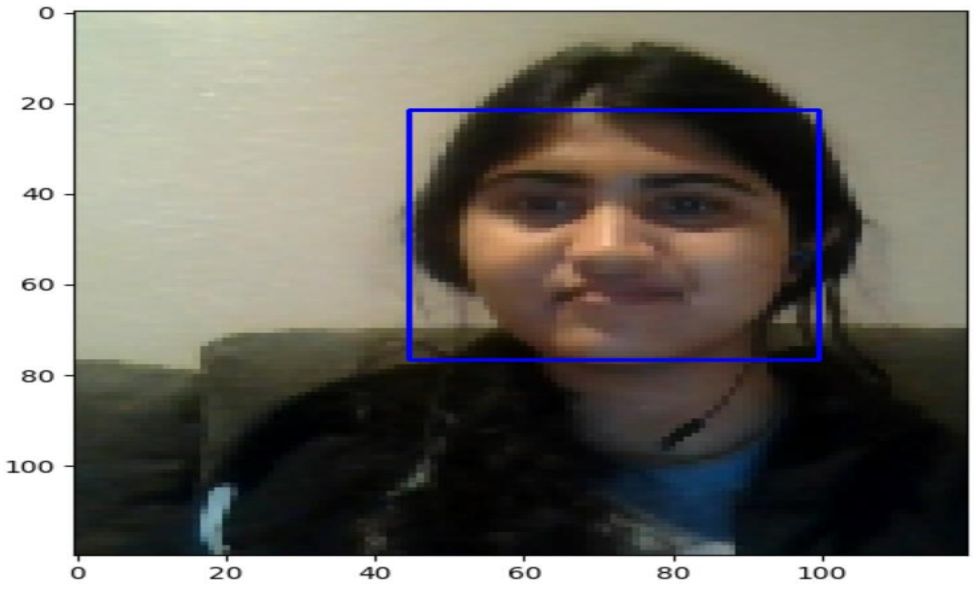
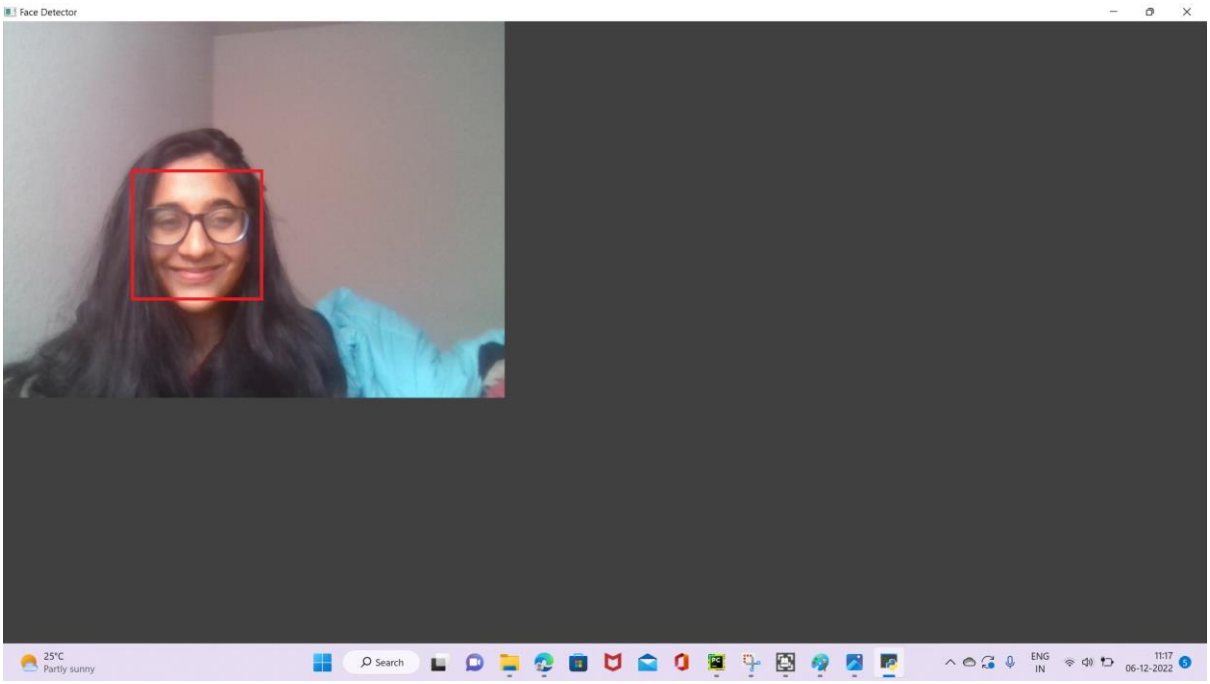


figure: Reduced Brightness in image

3. Face Detection



4. Real Time Face Detection



3.4 Analysis and Discussion:

Before implementing a neural network for face detection , we tried implementing a simple Haar Cascade Classifier and tested it with various parameters or arguments. The results in real time were precise in terms of regression or localization but the classification results were poor. The detection rectangle was drawn incorrectly around many objects in the frame.

As seen above this problem is no longer seen when we implement face detection using deep learning.

Unfortunately , the regression losses are not linearly reducing as we would have liked them to. This can be attributed to the size of the dataset. The model might be getting the same results for the images in the dataset and hence the repeated spike pattern as can be seen in the graph above.

The performance of this implementation could be made better by giving the model more number of images to train on.

But even with the current dataset , the results are much better in terms of classification than were observed with a simple classifier.

4.Conclusion

Face detection is significant when it comes to operations such as sentiment analysis, facial recognition and many more such operations. Face detection thus, can be achieved by implementing the following steps such as

1)Collecting and annotating image

2)Applying bounding box augmentation

3)Build a deep object detection model

4)Evaluating model performance

5)Detect faces in real time.

While the model could be improved in the future, at this point the model validates the presented concept.

5.Contribution

Tested face detection in images and videos to understand the simple Haar Cascade Classifier and observed its results.

Data Preparation and Cleaning

1.Installed the Dependencies and Setup the environment for the project.

2. Collected the images for the Dataset using OpenCV , annotated them , manually partitioned them into Train , Test and Validation data.

3. Performed Augmentation , Resizing on the images loaded the images in the tensorflow dataset.

4. Loaded the labels in the tensorflow dataset, combined images with their labels in the appropriate folders.

5. Plotted the performance results as a graph , analyzed the outputs.