FaceVision : Real-Time Face Detection From Video Streams

Aim → To built a Deep Learning model which will take input as video as input and draw a bounding box around face wherever the location of the face in video is.

Data Collection → We will require a high amount of data to train the neural network deep learning model. We have captured a total 384 photoes from laptop camera which is 1280-width and 720-height size. We will augment this data to create 30 times photes which we will store in the format of augmented versions of each image. Initially we have only 384 face images of captured from the laptop front camera.

Data Setting and Labelling of Images →

Dataset was uploaded to kaggle. This dataset was copied to the working directory of the kaggle notebook. Then we created "labels" directory to store the labels of each image. For labelling the face in images we used "labelimg" build library. References are as follows → Github link → ( ), (Video-link) → ( ). When I followed the video and on the labeling tool, it allows us to label the image with square bounding box around face which stores different information about face [object] in xml format. The output label of the image [image link] is given as [label link] in the xml format. We annected all the xml formatted labels into json formatted labels because we will be using Albumentations for a data augmentation. Then we created three foldees called "train", "test" and "val" and inside each of them we created images and labels foldees to store images and labels. (FOLDER STRUCTURE ON NEXT PAGE)

- kaggle working/
  - face_customer_image_dataset
    - Data (originally captured images)
      - Train
        - images
        - labels.
      - Test
        - images
        - labels
      - Val
        - images
        - labels.
      - Labels (xml formated labels)
      - labels (json formated labels)

Now, we will split the total data/images available in 70% for training, and 15% each for testing and validation. So we have 3~~68~~ 384 images, after splitting, we will be having 268 images in train, 57 images in test and 59 images in val. We will store these respective number of images in train/images, test/images and val/images respectively. Now we will have to store the json formated labels of each image in train, test and val in train/labels, test/labels and val/labels directory respectively.

Applying Image Augmentation on Images and Labels using Albumentations →

Image Augmentation → It is the process of creating new training images by applying random transformations to existing images.

Albumentation → It is a python library for fast and flexible image augmentation, this is especially useful in computer vision projects.

Then we defined a augmentation pipeline called "augmentor" using albumentations library which randomly alters images and bounding boxes to improve model generalization. Then we sanity checked whether image file is accessible and correctly read before doing further preprocessing like augmentation or training. We printed down the bounding box coordinates and class of the region. Now we will apply augmentation on train and validation data. To store this augmented data, we created one folder at → face custome imagedataset/aug-data and inside that created 3 folders train, test and val. And inside each of these 3, we created images and labels folders to store augmented images and their corresponding labels. Now we will apply augmentation on train, test and val data and store the images in aug-data folders along with their corresponding labels.
Then we resized the image to 120 * 120 pixels for standardizes input size to the neural network and normalized the pixel values to be in the range of [0, 1] by dividing by 255. This will give us all resized images in (120, 120, 3) with all pixel values scaled between 0 to 1.
Now we have to prepare the labels for augmented images, and we will do it with the help of loading class labels and bounding box for each image using its corresponding label file (.json).
Now we have train-image — ~~3840~~ ~~[illegible]~~ 30, train-labels test-image, test-labels ~~and~~ test-label and val-image, val-labels. We have to combine train, test and val image with train, test and val labels. We will shuffle the images and define batch size of 8.

Now before training the model and defining the model we will print/visualize the training image whether the class and bounding box is drawn properly or not.

Building Deep Neural Network Model using Keras functional API =>

Keras functional API => It is a way to build more flexible and complex neural network architectures compared to the simpler sequential API. Here API means Application Programming Interface provided by keras. We will use pretrained model of vgg16 which is pretrained on large dataset called "Imagenet". We will freeze the classification layer of vgg16 and use only convolution layers of vgg16 and we will add our custome classification and regression model to the output of vgg16 convolution layer. We have 2 problems to solve here, one is regression => Identifying the coordinates of the bounding box and another one is classification => Detecting whether face is available in the video/image or not.
Here vgg16 model architecture -> [Info].

Building custome CNN model =>

- Define input layer with shape of (120, 120, 3) because we have input pixels with 120 * 120 with RGB.
- Passed input layer through vgg16 imagenet pre-trained model with freezed classification layer.

→ Now we will define two different models, one for classification and one for regression. F1 is for classification and F2 is for regression

→ VGG16 output is passed through both F1 and F2

→ F1 output is passed through the class1 dense layer with 2048 nodes and relu activation function.

→ F2 output is passed through the regress1 dense layer with 2048 nodes and relu activation function.

→ class1 output is passed through class2 layer which is also the output for classification model.

→ regress1 output is passed through regress2 layer which is also the output for regression model

→ At the end, we will combine the output of both classification and regression model.

Loss function & Training of The model -

For fine-gained learning rate scheduling which will help the model converge more smoothly, we calculate learning rate decay per batch. We set Adam optimizer for training our neural network in Tensorflow with learning rate decay strategy.

For classification model, we will use loss function as binary crossentropy. We define localization loss explicitly for predicting where an object is located within an image. Predicting bounding boxes is not as a classification task, so it requires different loss fn than cross-entropy. Localization loss is a custom loss function that penalize that penalizes the model based on how far off its predicted bounding box is from the ground truth.

Now we will define custom keras model class FaceTracker, designed for a multi-task learning problem of face detection where we perform both classification and regression.

FaceTracker class → combines classification and regression in a custom model.

train_step() → custom training logic, forward pass, compute losses, backpropagate

test_step() → custom testing logic, forward pass and loss calculation.

compile() → adds support for custom losses (classification + localization)

call() → Defines model behavior when called.

Now after defining custom keras model class FaceTracker we will create an instance of the custom FaceTracker class by passing the facetracker model to it.
We will compile the model with adam optimizer with learning rate and decay, binary crossentropy loss for classification and localization loss for regression.
We will train the custome FaceTracker model using ".fit()" method for 10 epochs with "val" as validation loss and call-back.

Output Plots and Testing model on test data

We have drawn three plots after training the deep learning model and those are - total loss (VI) epochs, classification loss (VI) epochs and regression loss (VI) epochs. Then plotted the visualized the bounding box on test data.

Testing model on video stream:-

We captured a video of nearly 1 min and saved it in local machine. Then uploaded this video in kaggle data set directory. We passed this video frame through the model and detected faces in a video every 0.3 seconds, drawn a bounding box around them if detected, and display sampled frames with annotations.