OMAR ASSAADI

ECON 420

FINAL PROJECT : FACE DETECTION MODEL USING DEEP LEARNING

1- Introduction : What is it about ?

This project represents a deep learning application of convolutional neural networks (CNNs) used to detect faces in images and videos.

This model not only uses deep learning but also transfer learning which is a technique known in the Machine Learning and Deep Learning world. It has the capacity to apply acquired knowledge to new situations. For example , if a model is trained on a unique face, then with transfer learning it can recognize other faces. This is why I am using VGG16 which is a CNN used in transfer learning for classification and localization (regression). This one precisely is considered as one of the best vision model architectures for image recognition. Transfer learning initializes pre-trained weights from VGG16. This is what I have done in my model.

In this project I have been using Keras functional API in my Jupyter environment to load VGG16 and other important features. I also did classification and localization (regression) for the model to predict class labels and bounding box coordinates, and losses after. This is also called Fine-tuning.During the training process , my model predicts classification and localization (regression) losses with a certain function. Then we use an optimizer inside a model class, with a gradient to minimize the losses so that we achieve a high accuracy.

So what the model does is that it captures a video, in which it tries to locate a face with bounding boxes. These bounding boxes are an indicator of our model's data points generation.

PS: The code can only run in my laptop because it is done with Jupyter and with absolute paths that direct to my computer folders and data. I will then provide results in pdf of my code and a video that shows the results.

2- How did I develop this model

First of all this project has been done entirely in Python using a Jupyter environment.
I created this environment within Anaconda which is a software that lets you create Python environments that have some special Python packages or libraries.
I created an environment that had Tensorflow, Keras, Matplotlib, OpenCV, Labelme and Albumentations packages.

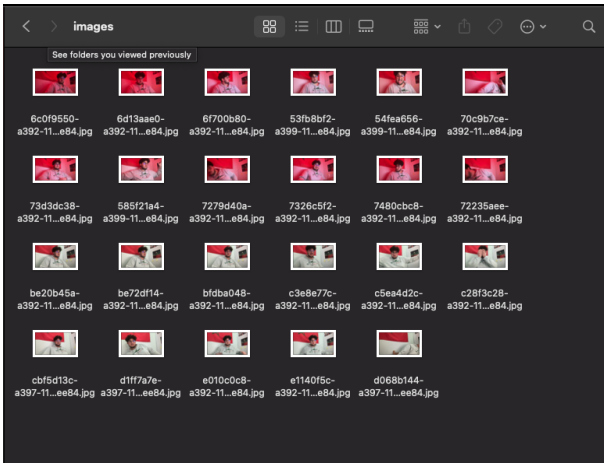
After importing these packages I created folder data, and inside it , a folder images where I would put all my image samples. To get these image samples, I used my own face and my own photos: for that, OpenCV helped me. With a simple code it can take a certain number of images and place it directly in your desired folder.

After that I had to label my face and create bounding boxes for my face: I used Labelme. Labelme is a software in which you can label everything you want for object detection. Then I formed a rectangle in every picture, which actually places 4 points , corresponding to the vertices of that rectangle, in JSON files. Each image had its own json file with the same name. I moved the json files in another folder called labels, still inside the data folder.

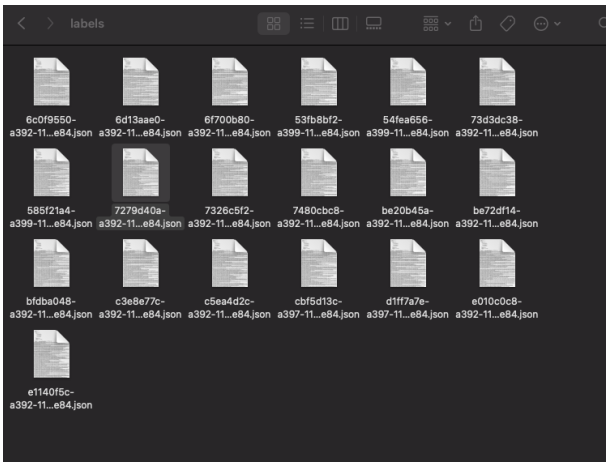
From this part we can start our project. We finally have a sample of data that we can augment to train afterwards. I used an augmentor with the Albumentations package , which helped me to create different instances of a single image. I tested it with one special image contained inside my training set , and it worked perfectly. After dividing the coordinates of the label by the height and width of the image, we were finally able to augment the image and the coordinates so that they fit perfectly after viewing the results.

This image shows us that the augmentor did its job and the labels perfectly bound my face.

The next part is to run an augmentation pipeline which actually does the same thing but for all the images multiple times. For this we need some training , testing and validation data so I manually and randomly moved 70 percent of the images to a training set, 15 percent to a testing set and the final 15 percent to a validation set.



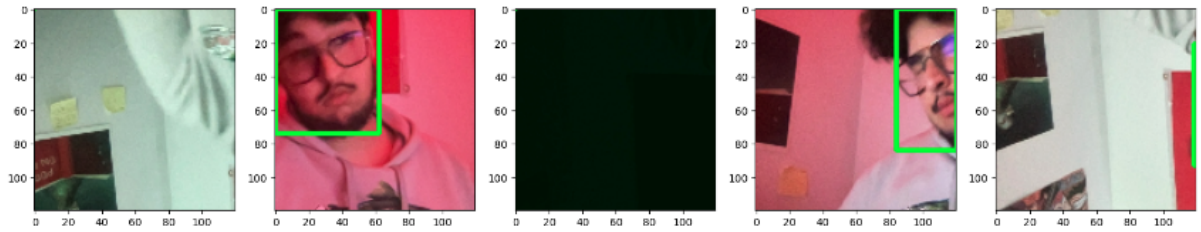
An example of my images in my test set.



An example of my labels in my test set.

Next, is the part where we run the pipeline. We have to go over all the three different sets , take each image and augment it multiple times. The same goes with its labels. All this goes into a new folder of augmented data where we have three folders train, test, valid within each we can find images and labels datasets.

After preprocessing each dataset we get a total of approximately 6300 images and labels for our training set, and 1300 for both the testing and validation set. To see if it worked properly let's generate 5 augmented images with their augmented label and see if it bounds my face.



As we can see the labels perfectly draw bounding boxes to my face and in the images where there is no face ,we see that there is no label.

The next step in the process is the neural network creation. For that we use VGG16 as stated in the introduction. So we create a function that will serve as a model to perform face classification and bounding box regression. VGG16 is used as a feature extractor for the classification and bounding box model. The whole idea of it is that VGG16 is passed through a pooling layer twice to get a dense layer for classification and twice again for bounding box regression. After this we return the two outputs with Keras. This represents my neural network creation.

Following , we have to create our optimizer and loss functions. I define an Adam optimizer with a very low learning rate as I did in my code. Afterwards, I create a function for classification losses and bounding box regression losses. I use the same formula as the famous object detection loss function YOLO. We get the distance between actual coordinate and predicted coordinate. Then we calculate the predicted and actual width and height of the box and we calculate the distance. The loss we get by adding both is what will help us in predicting coordinates and the size of the bounding box. For classification loss we just use the Binary Crossentropy loss function.

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

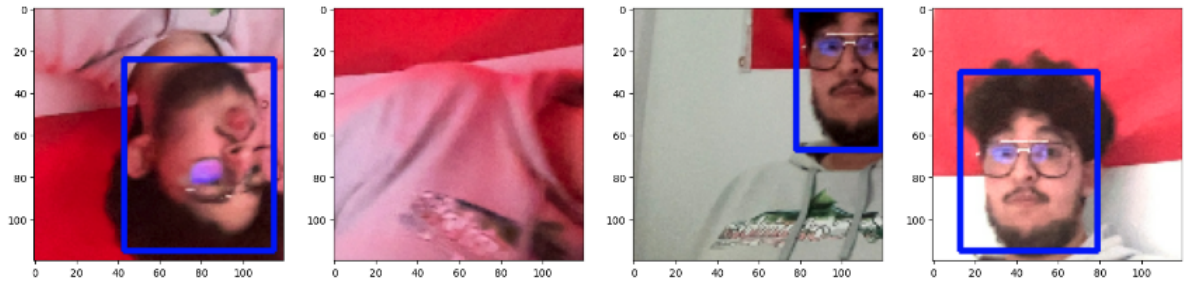
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

This is the formula we used for localization loss for our bounding boxes and sizes.

The final step is the one when we use the famous “model.fit” but before that we will create a class for our face detection model. This class is made of the classification and regression loss data. It has a train step and a test step. In the train step we calculate the total loss for a batch of data by doing the sum of classification loss of that batch and of the regression localization loss of that batch. We then use the gradient on total loss to reduce the total loss. To finish we use the gradient on the optimizer so that it minimizes the loss.
We do the same in the test step but without using an optimizer this time because we are actually testing it.

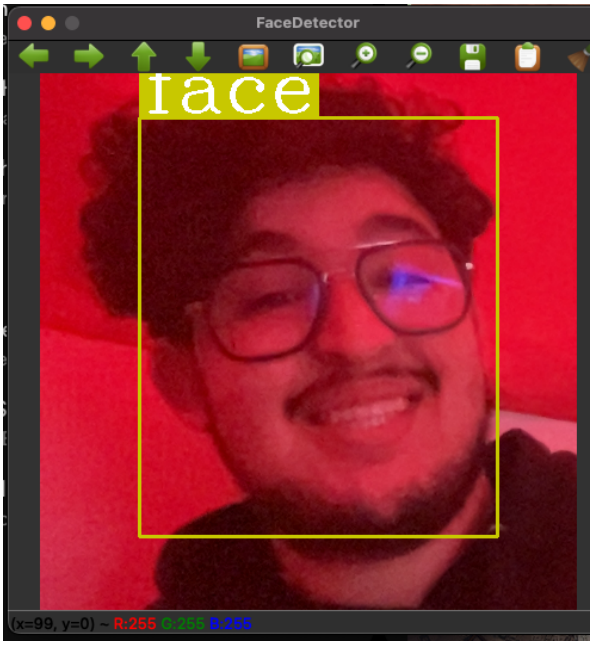
We finally create an object, we compile it and do some verifications and then we “model.fit” so we train our model with our training data and validation data with an epoch of 10 and we wait until it is finished.

At the end , we plot a graph to see if there are any discrepancies between training and validation and we make predictions in our test set to see if it works.



As we can see it works well in our test set.

The moment of truth is doing face detection in live, while our camera is open. Since everything went well, live detection must work. We use our saved pre-trained face detection model. If the predicted probability of face detection is greater than 0.9 we draw rectangles on the face. This is how I did my face detection deep learning model.



Face detection is working well.

3- Conclusions , Results , Limitations , Solutions

I think that my model works very well. It can recognize not only my face but other people's faces. What is incredible is that it can recognize photos of faces from a cell phone.

However my model has some limitations. It cannot recognise faces when we are just a little bit far from the camera. It still recognizes my face when I cover it with my hands. It may not recognize people of color.

For that I have multiple solutions. First of all, I should invest in a better camera and take more sample images but this time far from the camera. Secondly, I know that during the moment I was labeling my images, I was not only drawing rectangles on my face but I was also including my hair : this is why when I cover my face with my hands, the model still recognizes my hair. Finally I should get more dataset image samples including people of color and with different hair colors.

To finish, it was a great experience because I learned the process in writing a deep learning object detection model. Now I can do any object detection model of my choice, I just need to take good pictures of the object and augment it well.

Overall , I really liked this class and the professor. I can feel how machine learning is an amazing topic and I don’t regret taking this class !

Happy New Year !