Visual Recognition Mini-Project

Anish Rajan (IMT2018009)

Tanishq Jaswani (IMT2018077)

Manasa Kashyap (IMT2018040)

Aim: Build a vision based automatic door entry system, to allow door to open only if a human wear a mask.

Modules:

1. Human detection using YOLO
2. Face detection using Viola-Jones
3. Mask detection

Human Detection:

We used YOLO (you look only once) framework for human detection. It takes the entire image in a single instance and predicts the bounding box coordinates. The advantage of using YOLO is that it is extremely fast and can process 45 frames per second. We have used Darknet since it is fast and highly accurate.

1. We pass labeled data to the model. Out of all the classes, we choose only those belonging to the 'person' class.
2. If no human is detected in the image, it returns int.
3. YOLO takes the input image and resizes it into grids of S x S where S is some natural number.

In our code, prep_image() converts it into a tensor. This tensor gives us information about the coordinates of the bounding box and also the confidence score of all classes. Classes having score of less than 46% are eliminated.

4. Anchor boxes – they are a set of predefined bounding boxes which capture the scale and aspect ratio of the object class.

The algorithm predicts bounding boxes for each object. Output is something like

Object 1: (X_1, Y_1, $Height_1$, $Width_1$), $Class_x$

Object 2: (X_2, Y_2, $Height_2$, $Width_2$), $Class_x$

5. We pass the names of layers as parameters and get the index of output layers. We use Alex net with 2 fully connected layers.

So, here we had two options 1) to use predefined opencv.dnn YOLO function 2) to use and code GPU based YOLO algorithm. We initially used predefined opencv.dnn YOLO function. But it was lagging on the live videos. Like it was taking around 5 seconds to get the final output. So, we decided to shift to the pytorch GPU version of the YOLO.

For this we have referred some GitHub links and install Cuda and pytorch GPU enabled in our local machine. Initially, we face many problems in detection of GPU and dnn but at last we successfully make the laptop environment capable for this.

Following are the 5 .py files in our final code for the YOLO algorithm

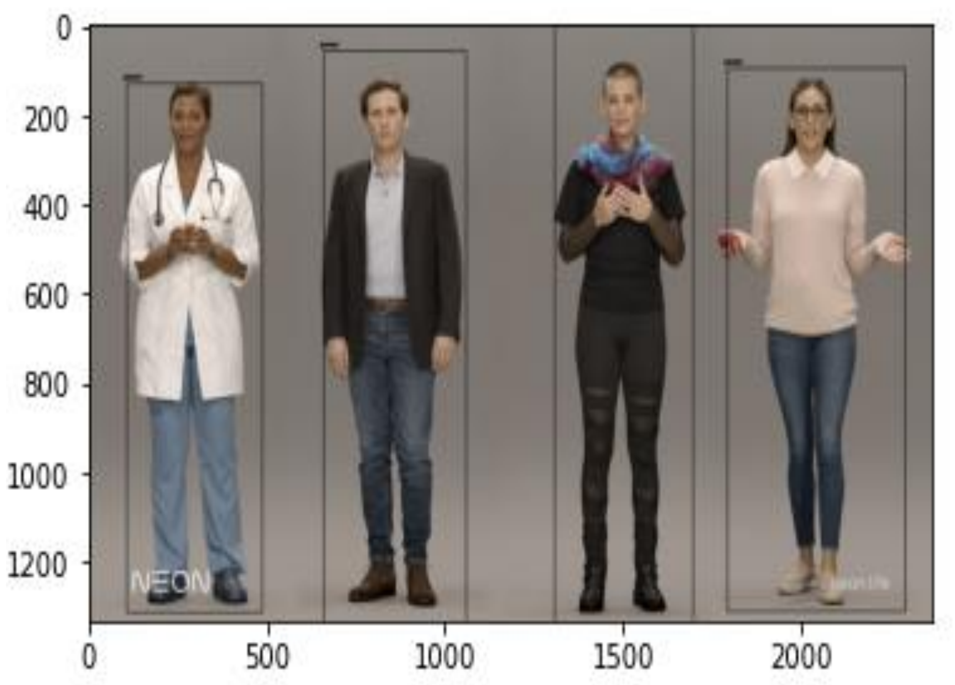
1. darknet.py: This is the official code for YOLO algorithm given by Darknet.
2. preprocess.py: This code is for converting the given image in the form of pytorch. Also, resizing the image keeping aspect ratio intact and padding the image for batch input.

3. detect.py: This code is for detecting the total images and passing each image to preprocess.py and after that passing them to person.py.
4. bbox.py: This part of the code is for getting the bounding box in the original image after removing padding and image aspect ratio.
5. person.py: This is the final code to combine all the function in the above 4 files and changing the final output in the form we want. So, in this code I am passing each frame of the video and after converting it to .cuda() I can use this image for the YOLO. As, our YOLO algorithm is running on the GPU I am converting each frame to the GPU. So, there is a function write() in person.py. As, in the Darknet there are 80 classes which the algorithm can detect and classify each object into one of them. For our project purpose we only want "person" class. This class has index 0 in the coco.names files (which Darknet was using to classify each object). So, in the write () function if the detected class of the object is not zero or height or width of the detected object is zero, I am returning (-1, -1), (-1, -1). Otherwise, I am returning tuple of upper left and lower right coordinates of the detected object. So, in the final.py code we are checking if coordinates are not equal to (-1, -1), (-1, -1) that means detected object is person and we will use that object for face detection which is second part of our project. Also, YOLO algorithm returns many bounding boxes around the object with different confidence. Here, confidence means that how much the algorithm is sure that it belongs to a class x. So, we have also applied Non max suppression to avoid multiple bounding boxes around the object which you can see in the below output image by our algorithm. So, after getting upper left and lower right coordinates of the bounding box I am cropping that image from the original image (current frame of the video) and passing them to face detection algorithm.

For example,

<u>Original image:</u>

<u>Detected Persons using our YOLO algorithm:</u>



There bounding boxes will be input to our Face detection function to detect face in them.

Face Detection:

We have used the Viola-Jones algorithm for real-time face detection in the image. Even though the training time is high, it is commonly used since it gives accurate results.

Two main stages in this algorithm are training, detection, adaptive boosting and cascading.

1. Training – this algorithm shrinks the image to 24 X 24 and looks for trained features within that image.

2. Detection – this algorithm is designed for detecting frontal faces. It converts the image to grayscale, detects the face in it, and then finds the location on the colored image. It outlines a box and searches for haar-like features. The box size can be set and it moves through every tile to detect a face.

Haar-like features have a dark side and a light side, and this is used to identify the features. Three types of features are

- Edge-features
- Line-features
- Four-sided-features

Every face has some similarities like nose region is brighter than the neighboring pixels and eyes region is darker than the neighboring pixels. Every feature has a value of its own, which we get by subtracting the white area from the black area.

Calculating the value of a feature is very difficult since there are a large number of pixels involved. We can do these calculations in a much easier way through integral images. Value of each box in the integral image is the sum of all boxes to its left.

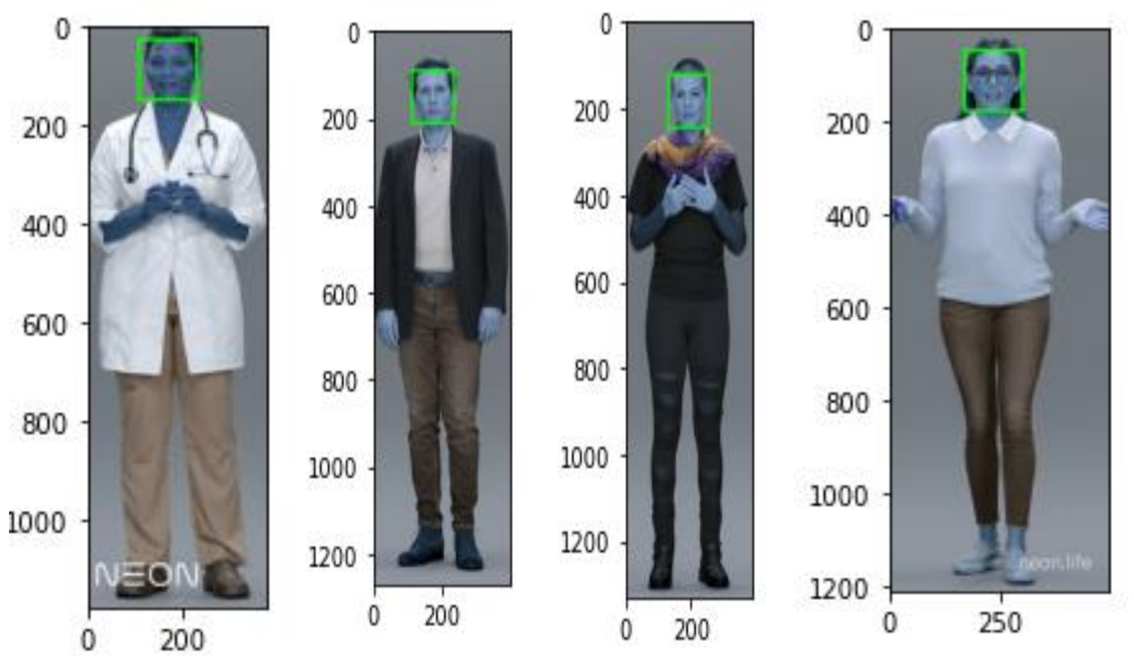
3. Adaptive boosting – we can get accurate results when we take into account all combinations of features. But this is very exhaustive. Adaboost chooses a second feature which best complements the current strongest feature instead of choosing the second strongest feature. This optimizes the algorithm and positives and negatives are given accurately.

4. Cascading – this is done to boost the speed and accuracy of the model. Face detection is divided into multiple stages. If the first stage thinks it might be a face the input is sent to the second stage, if not it is rejected. This continues until the last stage. If all the classifiers think it is a face, it is classified as a face and the output is given to the user. Else, it is rejected.

In our algorithm after applying YOLO on the current frame of the video it is returning coordinates of the detected persons in the frame. It is returning in the form of (x1, y1, x2, y2) where (x1, y1) are upper left coordinates of the bounding box and (x2, y2) are lower right coordinates of the bounding box. If YOLO does not detect any person, it will return an int data type which we are checking and in that case, we are returning an empty list. So, basically, we will return the list of all coordinates of the detected person in the form of (x1, y1, x2, y2).

Now, using that list (returned from YOLO) we are cropping the image and passing it to the predefined Cascade Classifier in the OpenCV. We are using the Haarcascade Frontal face Classifier to detect the face. After gray scaling the image, we are using this gray scaled image to detect the face. Using the .detectMultiScale() function of the Classifier we are detecting the face in the image. This function returns a list. Each element of this list is in the form of the (x1, y1, w, h) where (x1, y1) are the upper left coordinates of the bounding box and (x1 + w, y1 + h) are the lower right coordinates of the bounding box. So, at the end we are getting the coordinates of the bounding box of the faces in the cropped image of the person. So, this bounding box can be used to detect mask using mask classifier.

<u>Detected Face in each detected person:</u>



The bounding box of the above images will be input for the mask detection classifier.

<u>Mask Detection:</u>

For mask detection we had trained several types of models on different types of datasets, before concluding with one. The different types of models with the datasets that they were trained on are given below

<u>Model – 1(from Sir's reference with dataset from same github repo)</u>

The dataset used in this model had images for with and without mask. The problem with this dataset that we felt before only was that the images in this dataset were augmented with masks which we felt would confuse the model. Also, the images in this dataset were not of any other color except the surgical bluish white. This, we felt would confuse the model.

Model – 1 had the following architecture: -

- Convolutional Layer with 200 features and kernel size 3x3
- Relu Activation
- Maxpooling with kernel size 2x2.
- Flattening the output
- Dropout layer with probability of 0.5
- Dense Layer with output size of 50.
- SoftMax Activation

This model was trained for 20 epochs. We had plotted the loss curves for both the train and validation sets and unfortunately the model failed to converge. We had worked with different kinds of hyperparameters and different optimizers, but the model failed to converge.

The accuracy on the test set of the model was 44%. Then we tried and used this model on our real-time input in the final mask detection pipeline. The results were not very good. The classifier was unable to detect masks when the person was wearing a mask. We also tried thresholding the probability value so that we get better performance, but the results were not good, so we dropped this.

Also, another reason we felt that the model was not working was because the dataset had pictures were there was a large area except the face. The Viola Jones algorithm, however, only gives images where only the face area was being seen, which would have confused the model a little. So, we also started searching for new datasets.

Model – 2 Alexnet with New Dataset

First, we found a new dataset. This dataset was from Kaggle which was mask detection but had three kinds of classes where one class was wearing mask incorrectly. We though that this would be a good feature to add in out mask detection classifier.

In this one, we used a pretrained Alex net model from pytorch. The Alex net model gives 1000 features as final output. In this we tried 3 kinds of approaches: -

<u>1st approach:</u>

In this approach we had Alex net model and we added two layers at the end and the final model would look like this: -

- Alex net with output as 1000 features
- Dense Layer of output size 2 and input size 1000.
- SoftMax Layer

Also, in this model all the layers were trained not only the last 2 layers.

We trained this for 20 epochs. Though the model was working well on the training set but the performance on validation set was not good. The loss curves on the validation set were very unstable and did not converge well. The accuracy on the test set was 66%

There were 2 main reasons for this: -

- The model was getting confused between the classes of incorrect mask wearing people and people wearing mask.
- Again, the dataset did not have people with only faces and hence the model was not generalizable.

SGD with momentum was used for training but later we realized that Adam optimizer was working better with lr = 0.001.

When we used this model in our mask detection pipeline, the model was unable to detect masks again.

2nd approach

In this approach I removed images which were present with incorrect masks so that the model does not get confused.

Now we used the same Alex net model but added an extra Dense Layer of size 100 between the last layers. The final model looked like this: -

- Alex net with output as 1000 features
- Dense layer of output size 100 with ReLu activation
- Dense Layer of output size 2 and input size 100.
- SoftMax Layer

Adam optimizer was used for training with lr = 0.001

This was trained on the same dataset as earlier. The accuracy on the validation set was 70%.

But the model was again failing to detect masks in the images. Also, the model was giving different outputs on static videos where there was no movement.

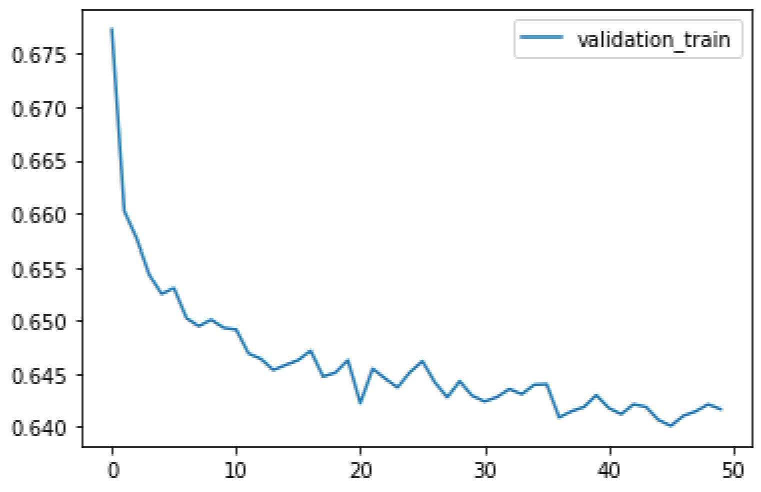
Another problem I saw was that the model was giving similar probability outputs for images in the dataset.

3rd approach

Now we used a different dataset. We found a dataset on GitHub wherein the user had used images where only the face region was present. The images were very similar to that of output given by Viola Jones Algorithm. We felt that this would be very good for the model to be trained.

In this case we removed the 100-size layer. Now, we also used Alex net as a feature extractor and used the 1000 layer as input to the last layer. The architecture looked like this: -

- Alex net as feature extractor. The requires_grad was set to False for all the layers.
- Dense layer of output size 2.
- SoftMax Layer



Adam Optimizer was used for training with a learning rate of 0.001.

This model gave an accuracy of 75% on the validation set. Though it looks like the accuracy is not good, it was trained on a greater number of images than the last dataset. The no of images was 3000 as compared to 1000 in the last dataset on which this was trained.

This model worked well on the mask detection pipeline but still it was not very good. During detection of people with masks, the model was giving very high probabilities to non-mask class and many times mask the number of times mask and no-mask classes were equal in a video. So, though this was best till now we had to think of something better.

Final Model

Now, we knew that the dataset was very good and only we needed to change the model. So, we tried different kinds of models, but Logistic Regression worked the best.

As in assignment 3b we used Alex net as a feature extractor and Logistic Regression model to train the classifier. Finally, the classifier looks like this

- Alex net with output size of 1000 which is used as feature vector
- Logistic Regression model for predicting mask or no mask.

The Logistic Regression gave us an accuracy of 97% on the validation set. This, we then tried with our mask detection pipeline. Initially, we did not get very good mask detection results.

Then we found a good optimal value of threshold for probabilities of mask detection. This was done by using different values from 0 to 1 with a step of 0.1. Trying this on the dataset gave us good accuracies and finally this was used in the final mask detection pipeline.

Model Performance and where the model fails.

Our model works well on inputs wherein the subject does not have much beard and the lighting is good. In the output videos that we have given, it is clear that the model gets confused when the subject has a beard. Also, we feel the model is biased towards males because the dataset had more male photos.

The model is unable to detect masks when there are female photos. Except this for people with long hair the face classifier fails to classify them as faces in most scenarios.

Conclusion

All the training and experiment were done on Kaggle GPU. Apart from this, 3 datasets were used, and we also learnt that the dataset matters in such tasks as the earlier datasets were not very good and the model was not generalizable in those cases. Finally, we found a good dataset and a mask detection pipeline was made.

Our model is very robust and works very good in Realtime also. The only problem was that the GPU in our laptops were not very powerful. Due to this, we did not get real time fast outputs and had to give videos to the model as the code was lagging and running slow (approximately 5 seconds lag).

We learn a great deal about how machine learning pipelines work and how human detections are done using YOLO Algorithm and RCNN.

References:

- https://github.com/ayooshkathuria/pytorch-yolo-v3
- https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/
- https://towardsdatascience.com/object-detection-face-recognition-algorithms-146fec385205
- https://pytorch.org/get-started/locally/
- https://varhowto.com/install-pytorch-cuda-10-0/
- https://download.pytorch.org/whl/torch_stable.html
- https://www.kaggle.com/pasqualedevita/facemaskdetection
- https://www.kaggle.com/wobotintelligence/face-mask-detection-dataset?select=train.csv