SIMPLE FACE DETECTION AND RECOGNITION SYSTEM

Derrick Lor
Miner School of Computer
Information
University of Massachusetts Lowell
Lowell, MA
Derrick_Lor@student.uml.edu

ABSTRACT

Computer Vision is a field of work specializing in video or image processing. Sub fields include detection, segmentation, classification and more. Face detection and recognition is a widely popular field and has been studied for many decades. The face is an important biometric feature of the human body, and it is innately used to differentiate between people. For this reason, the face has many uses, such as authentication, surveillance, and more [6]. This paper covers how computer vision techniques can be used to create a simple multi-classification face detection and recognition system from a directory of images using free libraries such as YOLO, OpenCV, and dlib. It is important to note that the multi-classification of this system is to match the name of a known face from a local database to the given input image. This is not a multi-task model, which extracts other factors such as race, age, or gender. Many advanced techniques have been developed over the past few decades, but simplicity is key, and some aspects may have been streamlined for a better understanding of the key principles. The key principles are detection, alignment, feature extraction, and comparison. It would be appropriate to state that detection and alignment are pre-processing steps. It can be done either preemptively, beforehand, or done in real-time, sequentially. Rather than stacking filter layers, as seen in Convolutional Neural Networks (CNN), the simple approach being used here is to use Histograms of Oriented Gradients (HoG) as the feature extractor. It can be seen from the evaluation portion of this paper that, the performance accuracy of this system is severely lower than that of other face recognition systems such as FaceNet, ArcFace, OpenFace, which is to be expected since those models use many more detailed feature extraction methods [5, 4, 1]. By the end of this paper, we would have built a primitive yet simple facial detection and recognition system using datasets and libraries that can distinguish between a recognized or unrecognized face stored within our database. Then evaluate it in the Celeb Faces dataset, achieving an accuracy of 20% to 30%.

1 INTRODUCTION

The motivation behind this project is to get a better understanding of the key underlying principles of facial recognition systems. By intuitively and simplistically implementing detection, alignment, extraction, and comparison, then we will have built the foundations to understand higher-level concepts.

2 RELATED WORK

There are many existing solutions and works related to the area of face detection and recognition. Related works such as FaceNet, ArcFace, and OpenFace have differing approaches to solving facial recognition. FaceNet (2015) uses a deep convolutional neural network to learn facial mappings in Euclidean space, then extracts facial embeddings to compare feature vectors [5]. ArcFace (2015)

proposed the use of Additive Angular Margin loss in the extraction process of face embeddings. This additional layer of feature extraction strengthens the model's ability to distinguish between faces, thus increasing performance on other deep networks [4]. The last network that is going to be mentioned is OpenFace (2016), and is going to be the model with most similarities to our simple facial recognition system. The OpenFace framework consists of taking inputs like image files, video, or webcam footage and automatically processes facial landmarks, alignment, histogram of oriented gradients, and dimensionality reduction through linear kernel support vector machines [1]. This is an oversimplification of the entire process; however, we will be borrowing some implementation details that will be revealed later.

From these basic synopses, and key ideas mentioned such as alignment, face embeddings, HoG, and feature vector comparison will lay the groundwork and foundation for our simple face detection and recognition system.

3 PROBLEM DEFINITION

The proposed system was created as a beginner study into the inner workings of face biometrics. A large and widely popular open-source library called face recognition has implemented many of the needed functions neatly packaged in a black box. A black box is a common term used to describe a mechanism that is concealed behind walls, to abstract inner workings. Simply put the needed data into the black box and wait for the resulting output. The end-user does not need to know a head of time the inner workings, if the desired output is what is needed. This project tries to make out the inner workings to be transparent and concise as possible.

Limitations and scope of this system and model are subject to the input images, dataset, feature extractor and distance metric used. The input image and datasets are assumed to have mask-free faces with minimal facial coverings. The datasets used to train the detection model were selected for its robustness to age, race, background, and gender. More information about the datasets will be explained in the experimental settings section of the paper.

One of the goals of this system is to have it run in real-time with webcam demonstrations. As well as performing with some degree of functionality and accuracy. The face alignment pre-processing part of the system was necessary to normalize all the face images to make it accurate in the latter part of the feature extraction process. The face landmark detector being used in the system is going to be relegated to the dlib library's "shape_predictor_68_face_landmarks.dat" model. This frees us from having to develop or train our own landmark detector, however we will still need to apply transforms to rotate and resize the resulting image.

There are many ways to go about comparing feature vectors, but in this simple model it is best to go with linear comparison methods. They're fast to compute and even simpler to implement. The distance metric used plays an integral role in model accuracy, for that reason we will implement three different distance metrics to be used: Euclidean, cosine, and correlation. The accuracy of these three metrics will be explained in the results section.

Another limitation is hardware. The most central part of the hardware devices being used is the NVIDIA RTX 4070 series GPU. The training of the face detector model with YOLOv11 was done locally. The limitations of the unified memory of the GPU were 12Gb. Meaning the model's parameters must be loaded into memory as well as all the memory requirements for each of the layers. This turned out to be a non-issue since we were using the nano checkpoint of the YOLOv11 model, which used approximately less than 3Gb of memory. This begs the question of why not use a larger model? The reasoning behind the nano model and not the medium model of YOLOv11 is due to simplicity. The nano model has fewer learnable parameters and fewer layers, which allows it to compute and train faster.

4 METHODOLOGY

The project makes use of some existing work such as YOLO, OpenCV, and dlib. The first two key principles are detection and alignment. To implement detection, Ultralytic's YOLO library will be used to train a face detection model. Then OpenCV and dlib libraries will be used to pre-process any detected faces and align them all to be normalized to a certain standard. Then the feature extraction

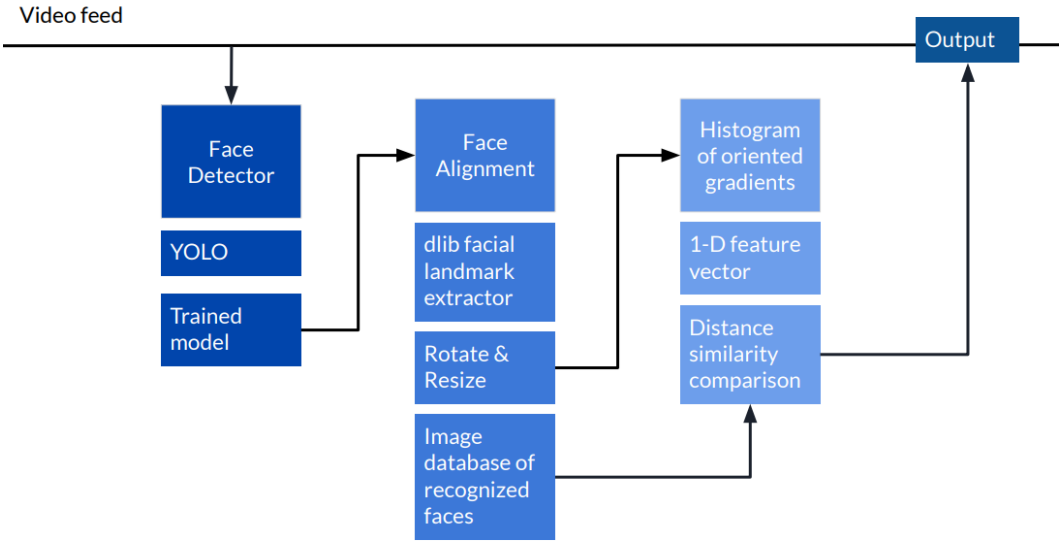

Figure 1: Architecture of proposed system.

Table 1: Hotkey Configuration

HOTKEY	DESCRIPTION
Q	Quit
R	Reload known faces list
F	Save detected face in whitelist folder
K	Save detected face in blacklist folder
F	Decrement distance threshold
X	Increment distance threshold
C	Change distance metric

portion of the process will be using scikit-image's implementation of histograms of oriented gradients. The final step is to do feature comparison by one-dimensional distance similarity by either using Euclidean, cosine, or correlation methods.

4.1 DETECTION

As previously stated, we will be using the YOLO framework to retrain a nano checkpoint model. The nano checkpoint model can be found from Ultralytic's YOLO documentation page [7]. With this framework we will load a pretrained checkpoint, pass the dataset configuration, and how many epochs to train, then model will automatically be retrained on the dataset. During the training process, all the performance metrics of the model will be saved and can be accessed for later use.

The OpenCV library, specifically for the python platform will be used. The purpose of using OpenCV is to utilize their image and video rendering loop, which will form the backbone of our facial detection and recognition system. Simply put, it allows the system to take video or image feeds and pipe it as input into our facial detection and recognition system. It does this in real-time with the ability to program certain key inputs. We will utilize this to allow certain keystrokes to provide extra functionality. Mainly, allowing the user to save detected faces, as well as other functionalities, such as changing the distance comparison methods, and the distance thresholds. Currently the hot keys can be found in Table 1.

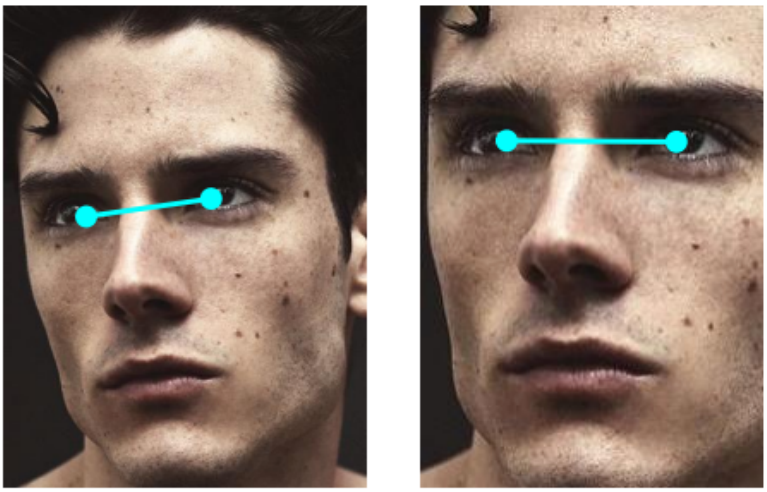

Figure 2: Image before face alignment (left). Image after face alignment (right).

4.2 ALIGNMENT

Next, the input images are fed to the face alignment portion of the system. As the name suggests, it aligns the detected face using dlib's facial landmark detector and applying matrix transformations. The purpose of the landmark detector is to highlight key features of the face. It allows for the detection of the eyes, nose, and mouth, but the system will focus solely on the eyes. Once the eyes have been located, the image must be transformed such that the eye level is flat and fixed to an absolute position. The transformation can be done using a single two-dimensional matrix transformation. First calculate the rotation needed to level the eyes. Then scale the image to fit into a fixed width and height dimension of 200x300. Then center the eyes onto this new reference frame. Finally, combine all these measurements into a 2-D matrix transform and apply it onto the input image. This pre-processing is done to all the incoming images in the input stream and the known faces database.

Consequently, the known faces database is a directory folder which holds the faces of people categorized into either the whitelist or blacklist. As the name suggests, labeled faces in the whitelist folder are people with given authorized access. On the other hand, labeled faces in the blacklist folder are people with unauthorized access. This includes people whose faces are not recognized by the simple face recognition system. As for the storage of the labeled faces, simply the basename of the image file will be used as the label and the extension ".jpg" or ".png" are acceptable file types, while other file types are ignored.

4.3 FEATURE EXTRACTION

The final step in the process is to run the histogram of oriented gradients feature extractor on the aligned image. The scikit-image library has the hog module which implements this for us. Simply pass the required number of orientations, pixels per cell, cells per block, and specify 1-D feature vector to be returned. Out of all the possible feature extraction techniques and methods available, the Histogram of Oriented Gradients (HoG) outperform existing features on human detection [3]. This paper [3] from Dalal and Triggs proved this experimentally through refining the binning, and normalization process. Although, they proved it worked exceptionally well in detecting humans, we can apply the same HoG process for detecting facial features like in [2]. HoGs are able to preserve and outline edge features. OpenFace does a similar HoG descriptor process to extract facial appearance features [1].

Once the 1-D HoG feature vectors have been extracted from the aligned image, it can be used to compare with other 1-D HoG feature vectors. Before any comparison can be made, the known faces

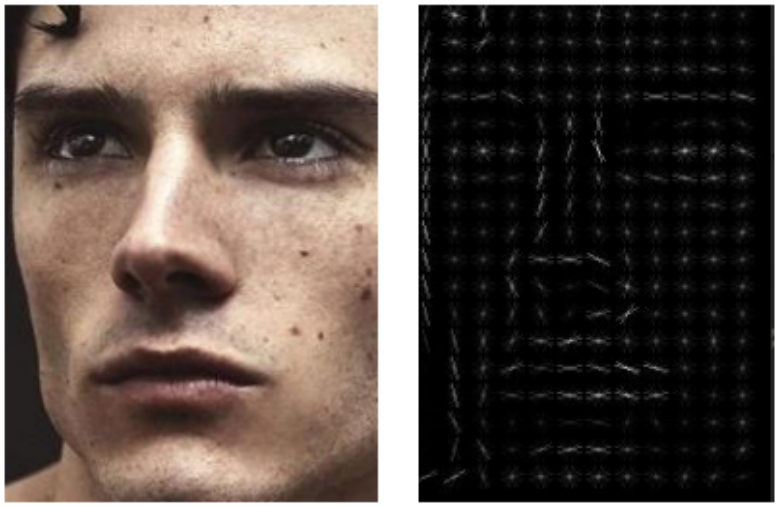

Figure 3: Image after face alignment (left). HoG visualization of image (right).

database must first be aligned and HoG features extracted. This happens at each start up of the system and is stored in memory.

4.4 DISTANCE METHODS

For the comparison methods, we have three available methods. Euclidean distance, which is the absolute difference between two values. Cosine distance, which is the measure of dissimilarity in the range between -1 to 1. Correlation distance, which measures the relationship between values that increase or decrease together.

- Euclidean distance$(\mathbf{A}, \mathbf{B}) = |a–b|$
- Cosine distance$(\mathbf{A}, \mathbf{B}) = 1 - \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \cdot ||\mathbf{B}||}$
  - $||\mathbf{A}||$ is Euclidean norm, L^2, of vector $\mathbf{A}$.
  - $||\mathbf{B}||$ is Euclidean norm, L^2, of vector $\mathbf{B}$.
  - Euclidean norm is square root of sum of all elements squared.
- Correlation distance$(\mathbf{a}, \mathbf{b}) = \frac{\sum_{i=1}^{n}(a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{n}(a_i - \bar{a})^2}\sqrt{\sum_{i=1}^{n}(b_i - \bar{b})^2}}$
  - a_i is element i of vector $\boldsymbol{a}$, with indexing starting at 1.
  - $\bar{a}$ is the mean of vector $\boldsymbol{a}$.
  - $\bar{b}$ is the mean of vector $\boldsymbol{b}$.

5 EXPERIMENTAL SETTINGS

5.1 DATASETS

This YOLO model was trained using multiple datasets. Namely the Face-Detect dataset and Faces dataset found on Roboflow Universe repository [8, 9]. The training, validation, and test splits vary. See Table 2 for the breakdown of splits for training, validation, and testing. The Face-Detect dataset was augmented with rotation and cropped images. Most of the augmentations were performed on the training split of images, therefore leading to an abnormally large training split while the validation and testing splits remained small. The Faces Dataset had no augmentation added to the dataset.

The original experimental settings had the use of a single training data to train the face detector model. However, during the training portion of the process, after about ten training epochs of the

Table 2: Dataset Information

Dataset	Train	Valid	Test
Face-Detect	2399 - 92%	100 - 4%	100 - 4%
Faces	1499 - 72%	313 - 15%	260 - 13%

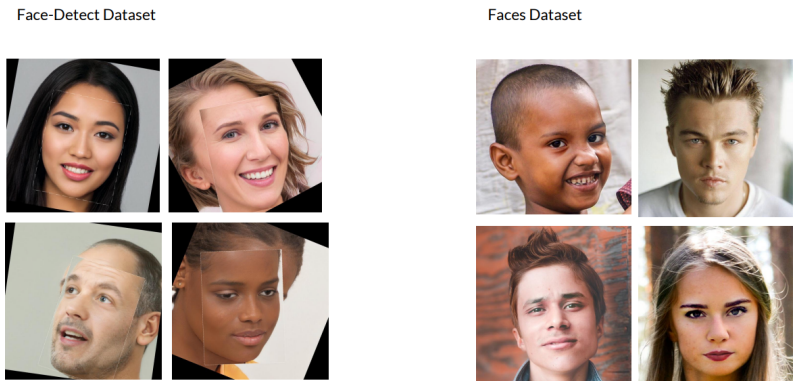

Figure 4: Sample images of Face-Detect and Faces Dataset.

first dataset, Face-Detect, the model's performance results looked promising with gradually decreasing training and validation loss with increasing precision, recall, and mean average precision at intersection of union thresholds. The training of the model could have continued from there, however when looking at a single minibatch validation run, it could be seen that easily identifiable faces were missing and, in some cases, had duplicate face detections.

Sample images of Face-Detect can be seen from Figure 4. Taking this into consideration and combining it with the fact that the training, validation, and testing splits were unorthodox, the conclusion was made to find and add another dataset which helped make the model more robust to background clutter and include more real-world validation and test splits. Hence in Table 2, we introduced the Faces dataset, where the training, validation, and testing splits were generally more acceptable by public standards of 70% training, 20% validation, and 10% testing split. This is a general rule of thumb, and small deviations from this percent split is acceptable.

Figure 4, shows sample images from the second dataset. Key characteristics from this new dataset include larger variations in image width and height dimensions, background clutter, depth of field, head angles, and lighting conditions. This new and more inclusive dataset would provide data augmentation in addition to the original dataset. These combined datasets would be used in conjunction with each other to train the YOLO model. All that was left was to include a new configuration file, that tells YOLO to use both datasets by combining the paths together.

5.2 PARAMETERS

Parameter settings were done using Adam optimizer with learning rate 0.002, momentum 0.9, and weight decay of 0.0005. As for the model, transfer learning was used to retrain an already existing yolov11 nano model with 100 layers and 2,582,347 learnable parameters. The total unified GPU Cuda memory used in training was 2.72Gb. The number of floating-point operations performed per second was 6.3 GFLOPs. Other than that, all the default values from YOLOv11 framework were used.

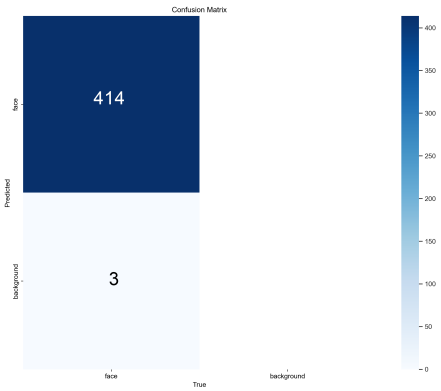

Figure 5: Confusion matrix of the trained face detector model.

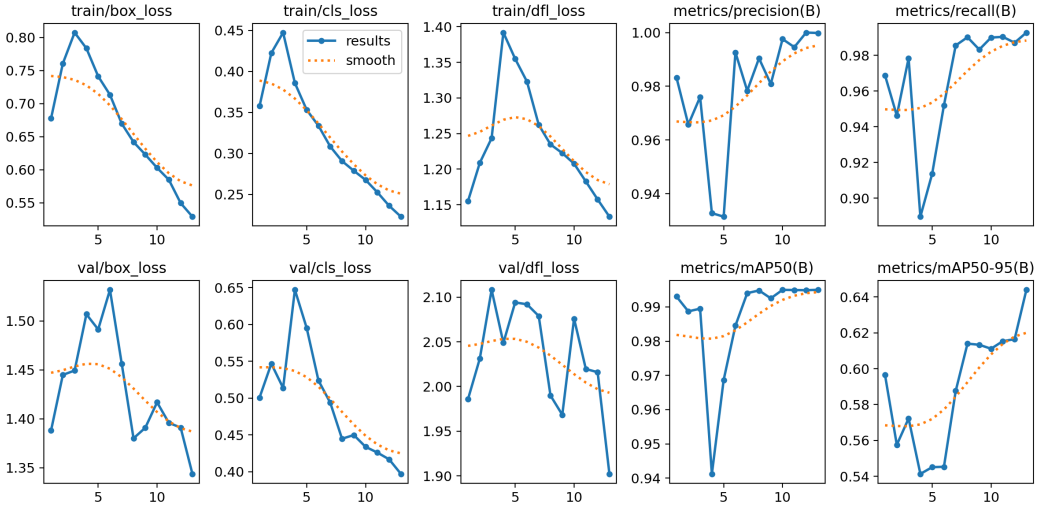

Figure 6: Results of last 13 epochs of training. 20 epochs of training total.

Figure 7: Sample images of Celeb Faces Dataset.

Table 3: Average Accuracy on Celeb Faces

Distance Metric	**Average Accuracy**
Euclidean	0.198
Cosine	0.207
Correlation	0.345

6 RESULTS AND ANALYSIS

6.1 DETECTION MODEL

In this section of the paper, there will be various quantitative metric measurements that will be used to evaluate the training performance of the face detection model. Some of these metrics include accuracy, loss, precision, recall, mean average precision, etc. Notable we will be focusing on precision and accuracy of the model. That is not to say the other metrics are not noteworthy, but are usually less recognized as good indicators. After each epoch, generally if the model's loss is decreasing and the accuracy is increasing, then the model can still be improved. The face detection model was initially trained for 7 epochs as a sanity check for good convergence. Once that had been verified through training performance, the model was allowed to train for 13 more epochs for a total of 20 epochs. After the 20th epoch, the model's validation performance proved worthy and did not need any more training. Suffice it to say, the detection portion of the face detection and recognition system was done.

The results of the face detector model can be seen in Figure 5 and Figure 6. Figure 5 shows the confusion matrix after the final 20th training epoch. From it, we see 414 correctly predicted faces and 3 incorrectly predicted faces. This gives an accuracy score of 0.992. An almost perfect score. A total of 20 epochs were used to train the model, however the graphs from Figure 6 only show the last 13 epochs. Although this is not the full graph, we can still quantize and summarize the results of the training. The blue line describes the actual metrics measured and the yellow dotted line represents the metrics when applying a smoothing factor. Qualitatively speaking, one of the notable features is the sharp increase in loss and a corresponding decrease in accuracy and precision during the first five epochs in the graphs. Not to worry, since the weights of the model were initialized with random weights. This is to be expected in the beginning performance as the model's weight vectors are being corrected to minimize the loss and update according to direction of the gradient descent. To generalize, the training loss after each epoch follows a downward trend. This means that the model is back-propagating and updating weights to minimize the loss and increase accuracy. To back this up, the precision and recall graphs show a dramatic increase after the fifth epoch and stabilize after the 10th or so epoch. Without a doubt if the model kept training, then the performance would keep increasing. However, since the face detection model is only one part of the face recognition system, the current state and performance of the model is satisfactory.

6.2 ALIGNMENT AND FEATURE EXTRACTOR

The next topic to discuss is the evaluation of the alignment and feature extraction process of the system. In the architecture found in Figure 1, the output of the face detection model is directly piped into the face aligner and then subsequently into the HoG feature extractor. So, to evaluate the performance of the alignment and feature extractor portion of the system, there must be a basis to be compared to. Like how the face detection model compares against a ground truth bounding box with the model's predicted bounding box. We must provide a labeled face and compare it against a set of labeled faces in our database directory. Introducing, the Celeb Faces Dataset with 31 different classes of people [10]. After cleaning the Celeb Faces Dataset, we end up with 29 names of celebrities. Samples from the Celeb Faces Dataset can be seen in Figure 7. To expand, each of the named celebrities' classes will have multiple images of differing angles, lighting conditions, image resolutions to compare against.

To test and evaluate the alignment and feature extractor, we will try to replicate the use case scenarios. Wherein a single face image of each person is stored in our known faces database, then a random image from the Celeb Faces Dataset will be used as input through our simple face detection and recognition system. Theoretically, the system will output a matching name that corresponds to the ground truth label. If correctly predicted, then accuracy of the system increases, if not, it will count as a failure. Since our system allows for the use of three different distance metrics, Euclidean, cosine, and correlation, we repeat this batch evaluation process multiple times and average the results. Table 3 gives the average accuracy of each of the distance metrics over 10 batches, each with 10 experiments.

Possible reasoning behind the lackluster results may be due to a variety of factors. It can either be due to the system architecture or the evaluation dataset being used. To address the system architecture first, the biggest problem may lie with the feature extractor. HoG are known to be used as an edge or contour detector. If the image itself has no recognizable landmarks with sufficient gradients, then the HoG binning process might overlook and discard small facial feature data. To play devil's advocate, maybe the problem lies in the input image itself. The known faces database assumes to have a clean, forward facing, neutral emotion, and timeless face image of the person. In the real world, time is ever changing and the face changes with it. Facial hair may hide contours or make new ones. Perhaps eye health is deteriorating, and glasses are now being worn. The possibilities that need to be accounted for and precautions to look out for are too many to handle for this simple system architecture.

7 FUTURE WORK

This simple face detection and recognition system is still a far cry from the latest technological developments of facial recognition. As seen from the evaluation accuracy, there is still a large room for improvement. Some key areas that could use further growth and refinement are the detection model, face alignment, and feature extraction portion of the system.

7.1 DETECTION

Although the current face detection model looks complete, perhaps changing the perspective or angle of attack may prove to be better. The YOLO framework is not just limited to detection and bounding boxes. Another mode or task that YOLO is accustomed to is segmentation. The segmentation task is another step beyond just detection. Not only does the model have to identify the object of interest, but it must also output the outline of each object. Ultimately, the model learns the image features that are important and only outlines the intended objects to be segmented, throwing away any unnecessary details. This extra step of drawing masks allows for a clean contour and outline of the object of interest, effectively removing any background. As you can imagine, this would be an improvement over the current detection system, which only draws boxes over the object of interest. The downside of the detection model is that backgrounds may be included in the bounding box. Which is later piped into the alignment portion of the system. Although the alignment tries to crop and resize the image to fit much of the face, some background edges can be seen in the intermediate steps.

7.2 FACE ALIGNMENT

The current face alignment system takes the incoming input image and applies rotation and scaling 2-D matrix transforms. This manipulation of the image data works with either images or videos. However, only leveraging 2-D transformations on a 3-D object leads to slight miscalculations due to imperfect pitch, yaw, or roll. OpenFace's approach to solve this problem is to internally build a 3-D model of the face using the image data as projections [1]. This allows them to utilize video sources to feed and create a 3-D model from 2-D sources. In turn, the 3-D model more closely represents the facial features of the person and captures depth information that is not reproducible in our simple face recognition system.

7.3 FEATURE EXTRACTION

Our system utilizes only a single feature extractor or layer to create a face embedding. On the other hand, FaceNet uses a 22-layer deep model with large convolution activation maps with pooling [5]. This allows FaceNet brake down facial features and condensed them into a single 1-D 128 length face embedding. By utilizing the deep model approach, they were able to achieve accuracy up to 99.63%. Needless to say, our model achieved accuracy up to 34.5%. A less-than-ideal model. So, the next logical improvement is to deepen the feature extraction portion of our system. Not as deep as FaceNet, but deep enough to push the accuracy scores upwards to above 70% would be reasonable.

8 CONCLUSION

In conclusion, we have built a simple multi-classification face detection and recognition system using open-source libraries like YOLO, OpenCV, dlib, and scikit-image. This system prioritizes simplicity and core principles of face detection, face alignment, feature extraction, and feature comparison. Detection with the use of the YOLO framework, face alignment with dlib, and feature extraction with Histogram of Oriented Gradients. Evaluating this multi-classification system on the Celeb Faces Dataset yielded performance accuracy of 20% to 30%. This is significantly lower than other notable state-of-the-art models like FaceNet, ArcFace, and OpenFace. However, with future work and improvements, this system may achieve greater heights.

9 REFERENCES

1. Baltrusaitis, Tadas & Robinson, Peter & Morency, Louis-Philippe. (2016). OpenFace: An open source facial behavior analysis toolkit. 1-10. 10.1109/WACV.2016.7477553.

2. Cai, Z. Yu, P. Liang, Y, et al. "SVM-KNN Algorithm for Image Classification Based on Enhanced HOG Feature," International Conference on Intelligent Systems and Image Processing (2016).

3. Dalal, N. and Triggs, B., "Histograms of Oriented Gradients for Human Detection," IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, San Diego, CA, USA.

4. Deng, J. Guo, J, et al, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," JOURNAL OF LATEX CLASS FILES, VOL. 14, NO. 8, AUGUST 2015.

5. Schroff, F. Kalenichenko D. Philbin, J. "FaceNet: A Unified Embedding for Face Recognition and Clustering", In: CVPR (2015)

6. Stan Z. Li, Anil K. Jain, Jiankang Deng, "Handbook of Face Recognition The Deep Neural Network Approach," 2024, Springer Edition.

7. Ultralytics. (2025, February 26). YOLO11 NEW. https://docs.ultralytics.com/models/yolo11/

8. FaceDetect Object Detection Dataset by HuanHoaHoe. (2022, September 12). Roboflow. https://universe.roboflow.com/huanhoahoe/facedetect-jb2ph

9. Faces Object Detection Dataset by school. (n.d.). Roboflow. https://universe.roboflow.com/school-7y83u/faces-ylez0-9vot8

10. Face Recognition Dataset. (2020, November 6). Kaggle. https://www.kaggle.com/datasets/vasukipatel/face-recognition-dataset