

Real Time Face Detection and Recognition

Pragati S.Kulkarni¹, Dr.Mahesh S.Kumbhar²

¹*Electronics Engineering, Rajarambapu Institute of Technology, Islampur, Maharashtra, India*

²*Electronics Engineering, Rajarambapu Institute of Technology, Islampur, Maharashtra, India*

Abstract— Face detection and recognition based applications are widely used today. These applications are extensively utilized in many sectors. Major difference between face detection and recognition is the former only finds and detects individual face in an image whereas later system is able to classify the face from a sample of different faces. This project focuses on the implementation of both a face detection and recognition system in MATLABR2023b. Different computer vision toolboxes have been used in the program. In this work the Deep Transfer Learning method using AlexNet pre-trained CNN is proposed to improve the performance of the face-recognition system even for a smaller number of images. The transfer learning method is used to fine-tune on the last layer of AlexNet CNN model for new classification tasks. All the experiments were tested on real-time database of different human faces with different label classes. As a result, accuracy is highly improved.

Keywords—Face detection and recognition, database, MATLAB, CNN, AlexNet, Transfer Learning

I. Introduction

Faces play an important role in human interactions. Now a day's face is used as a biometric identifier in many applications like access control for security, criminal identification, surveillance and many other commercial applications. Face detection is a procedure by which one can extract the facial region of humans from the images. This technology has long history but still it is a part of research and challenging biometric technic. This technology has been developing from conventional method to emerging technic. M. D. Kelly et al. (1971), had conducted the research on the face recognition and the first face recognition algorithm was develop in early 1970s [18]. In the late 1980s, the development of computer technology and optical imaging technology was improved and the real entry into the application phase of face recognition in the late 1990s. In early research on face recognition, the researcher mainly focuses on methods called geometry methods used to match simple features with image processing techniques [17]. Later, the holistic method such as principal component analysis (PCA) and linear discriminant analysis (LDA) appeared and become popular [19]. The ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) was held annually starting in 2010, after the birth of the large-scale dataset, ImageNet[19]. Recently CNN models are used to achieve good results but the main problem is that it required large labeled data might be difficult to collect. Also we don't have capable hardware to train this data. As a result, a CNN (AlexNet) based deep transfer learning is proposed. It has popular technic that knowledge from related task transfer to new task which reduced training time and high accuracy achieved even with small datasets

II Methodology

The evaluation of Real-time Face detection using viola-jones algorithm and Recognition system with CNN(AlexNet) pretrained network with transfer learning technic will be presented in this work.

A. Software

Face detection and recognition system implemented in MATLABR2023b platform. In MATLAB there are many toolbox functions are available as source. Deep Learning Toolbox Model for

AlexNet Network, Computer Vision Toolbox, and Parallel Computing Toolbox™ are the toolboxes used in this proposed system.

B. Databases

We have used real-time captured images through program controlled webcam of two different human faces namely Pragati and Aniket respectively each of 150 images of 640x480 pixels afterwards it will resize to 227x227. Database generation takes place with turning ON webcam. Webcam captures images and detect where face is present using Viola-Jones Algorithm. It captures the images up to the set value for different person faces.

1 Face detection methods there are various algorithms for face detection. These algorithms are broadly classified into two categories

1.1 Feature based methods

The algorithms present in this category detect faces based on some simple features present in the facial regions. They do not take into consideration the effect of ambient light, rotation and pose. One of the widely used methods in this category is based on the skin colour model. Here the image is segmented and the face is determined by the probability of finding skin colour in the segmented regions [8,3].

1.2 Learning based methods

The algorithms belonging to this category are based on statistical models and machine learning algorithms. These algorithms are robust and require greater computational time than feature based methods. They give good results for different rotational poses even in poor lighting conditions. Hence, such methods are more preferred than the feature based methods [1].

2 Viola Jones Algorithm

This is a learning based algorithm which is used for object detection [4] The cascade object detector uses the Viola-Jones algorithm to detect people’s faces, noses, eyes, mouth, or upper body[10]. The Viola-Jones face detection method is divided into three main parts (Integral image, classifier learning with AdaBoost and cascade structure) that make it possible to build a successful face detection that can be used on real time application.

Fig 1a shows original image. Rapid computation of Haar-like features using the integral image as shown in fig 1b and its formation shows in fig 1c. It uses Haar features and a cascade of classifiers to identify the objects. The Haar features are computed by using the integral image. The best features are chosen by using the Adaptive Boost (Adaboost) algorithm. This procedure takes place in every stage and there is a cascade of such stages. In every stage, the incorrectly detected faces are discarded. Thus higher the number of stages better will be the accuracy of face detection.

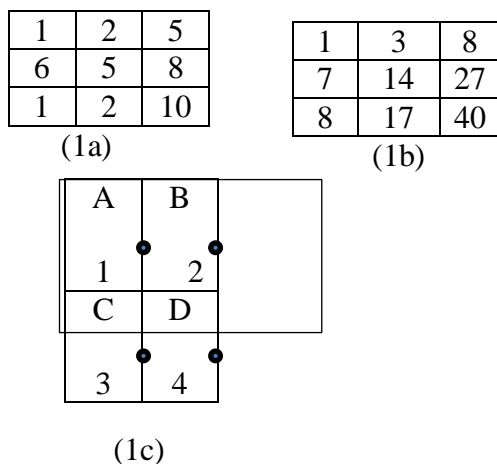


Fig1a. Original image
 Fig1b. Integral image
 Fig1c. Integral image formation

2.1 Haar Features

The entire image is divided into small windows or rectangular regions of size $M \times M$. The features are calculated for each window individually. Generally, three types of features are employed for face detection viz. two-rectangle features, three-rectangle features and four-rectangle features. The two-rectangle feature is the difference between the sums of the pixels within two rectangular regions. These rectangular regions are of same shape, size and are adjacent to each other horizontally or vertically. A three-rectangle feature calculates the sum of pixels within two outside rectangles and is subtracted from the sum of pixels in the center rectangle. Finally a four-rectangle feature computes the difference between diagonal pairs of rectangle. These features are shown in Fig. 2. They are also called as weak classifiers. The three rectangles Feature shown on the face in the detected face denote that the eyes region is darker than the nose bridge.

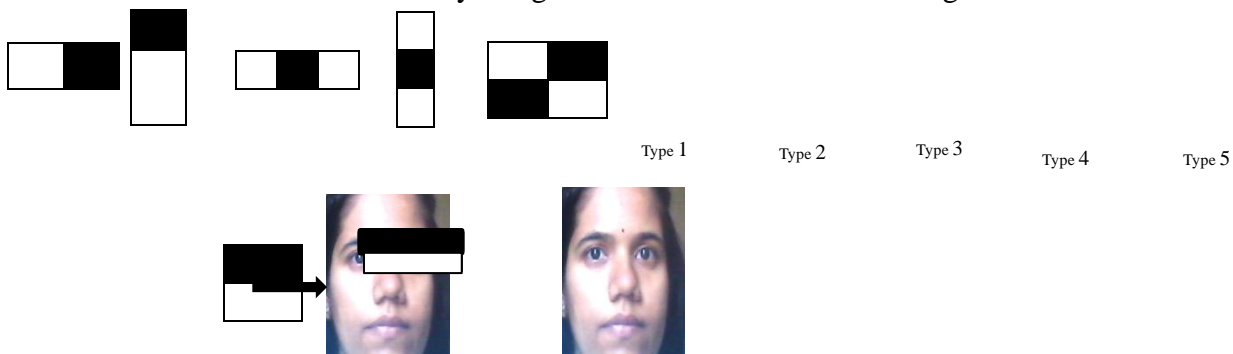


Fig. 2 Haar features

The second feature on the face is based on the intensity difference between the eye region and the upper part of the cheeks.

B. Integral image formation Haar like features are computed very rapidly using an intermediate representation for the image - integral image [5]. The formula for calculation of integral image is shown in equation 1.

$$p(x, y) = \sum_{i=1}^x \sum_{t=1}^y l(i, t); 1 \leq x \leq M, 1 \leq y \leq M \quad (1)$$

Here p is the integral image as shown in Fig. 2. 'i' is the original image. M is the size of the rectangular region. The integral image at location 1 in Fig. 1b is the sum of pixels in region A; at location 2 it is sum of pixels in region A+B; at location 3, it is sum of the pixels in region C+A; and at location 4, it is sum of the pixels in the region A+B+C+D.

2.2 Cascade

After selecting the best features in every window, we have to now decide which of these windows contain faces. On an average, only 0.01% of all windows in an image are positive i.e. they contain faces. To find the positive windows the initially detected faces have to pass through a number of cascaded stages. Every stage reduces the number of false positives i.e. the regions that are incorrectly detected as faces.

Every stage is a classifier that is built by using some features. More and more features are added in every subsequent stage, thus making the classifier more complex. In every stage the detected region can either be rejected or passed on to the next stage. Thus, only the region which successfully crosses all the stages is classified as a face. This procedure is shown in Fig. 3. Higher the number of stages in the cascade classifier better is the accuracy of detection. But this comes at the expense of computation time. Higher the number of stages in the classifier more is the computation time. Hence, there is a trade-off between accuracy and

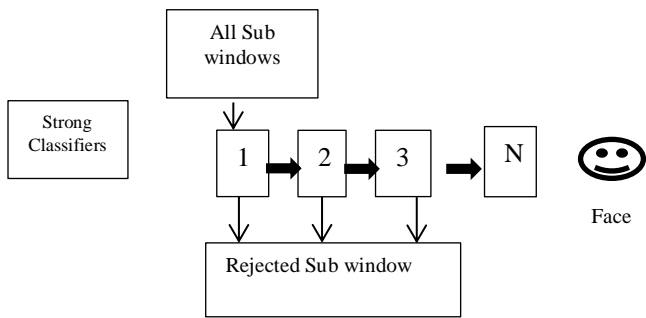


Fig 3 Cascade structure

2.3 Trained cascade classification model

Trained cascade classification model, specified as a character vector in Table no 1. The Classification Model property controls the type of object to detect. By default, the detector is configured to detect faces. You can set this character vector to an XML file containing a custom classification model as describe in fig, or to one of the valid model character vectors listed below. You can train custom modification with the ‘traincascadeobjdetector’ function. The function can train the model using Haar-like features, histograms of oriented gradients (HOG), or local binary patterns (LBP). [10]

Classification-on Model	Image Size Used to Train Model	Model Description
'FrontalFaceART'(Default)	(20, 20)	Recognizes faces that are forward-facing and erect. This model is based on the classification and regression tree analysis (CART) and consists of weak classifiers. Haar characteristics are used by these classifiers to encode facial features. It is possible to model higher-order connections between facial features using CART-based classifiers. [12]
FrontalfaceLBP	(24, 24)	Recognizes faces that are forward-facing and erect. This model is made up of decision stump-based weak classifiers. These classifiers encode facial features using local binary patterns (LBP).

		Robustness against variations in illumination can be offered by LBP characteristics.[13]
'UpperBody'	(18 22)	Detects the head and shoulders region, which is known as the upper-body region. This model encodes the intricacies of the head and shoulder region using Haar features. This model is more resilient to changes in position, such as head tilts and rotations, because it makes use of more features in the head region. [14]
'EyePairBig' 'EyePairSmall'	(11 45) (5 22)	Recognizes two eyes. A reduced image is used to train the 'EyePairSmall' model. Because of this, the model is able to identify smaller eyes than the 'EyePairBig' model.[15]
'LeftEye' 'RightEye'	(1 2 18)	Detects the right and left eyes independently. These models consist of decision stump-based weak classifiers. Haar features are used by these classifiers to encode information.[15]
'LeftEyeCART' 'RightEyeCART'	(20 20)	Detects the right and left eyes independently. These models consist of CART-trees, which are weak classifiers. CART-tree-based classifiers are more adept at modeling higher-order dependencies than decision stumps. [16]
'ProfileFace'	(20 20)	Recognizes profiles of upright faces. This model is made up of decision stump-based weak classifiers. Haar features are used by these classifiers to encode facial information.
'Mouth'	(15 25)	Finds the mouth. This model is made up of weak classifiers that encode mouth details using Haar features and are based on a decision stump.[15]
'Nose'		

	(15, 18)	This model is made up of weak classifiers that encode nose details using Haar features and are based on a decision stump. [15]
--	----------	--

Table 1 Trained Cascade Classification Model

2.4 Result of Database Generation



Fig 4 Class 1 Database labelled as Pragati



Fig 5 Class 2 Database labelled as Aniket

C Training of Data

Training data is the data you use to train an algorithm or machine learning model to predict the outcome you design your model to predict. Training data are collection of samples that are used to teach or train the model .training datasets are used to understand the patterns and relationship within data, so learning to make prediction or decisions without explicitly programmed to perform a specific task. Here I have used convolutional neural network (CNN) with pretrained (AlexNet) transfer learning model for training of data

3.1 Convolutional Neural Network

The convolutional network consists with different layer of artificial neurons. Artificial neurons are being defined by the value of its weights. When being fed with the values (of the pixel), the artificial neurons of a CNN recognizes various visual features and specifications.

3.2 Transfer Learning Using AlexNet

This example shows how to fine-tune a pretrained AlexNet convolutional neural network to perform classification on a new collection of images. utilizing a neural network to classify a fresh set of photos After being taught on more than a million photos, Alexnet is able to identify 1000 different object categories,. For a variety of photos, the network has learned rich feature representations. As seen in fig. 6, the network receives an image as input and produces a label for each object in the image along with the probabilities for each object category. In deep learning

applications, transfer learning is frequently utilized. A pretrained network can be used as a foundation for learning a new task, and fine-tuning a network using transfer learning is typically far quicker and simpler than training a network using

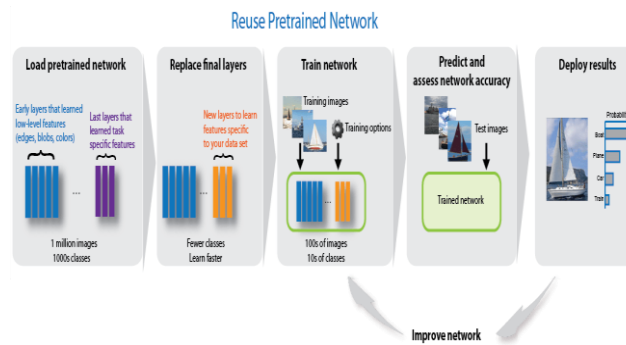


Fig 6.Reuse Pretrained Network[10]

3.3 Load Data

Unzip and load the new images as an image data store. Image Data store automatically labels the images based on folder names and stores the data as an Image Data store object as shown in fig 7. An image data store enables you to store large image data, including data that does not fit in memory, and efficiently read batches of images during training of a convolutional neural network [10].

Result of Load Data

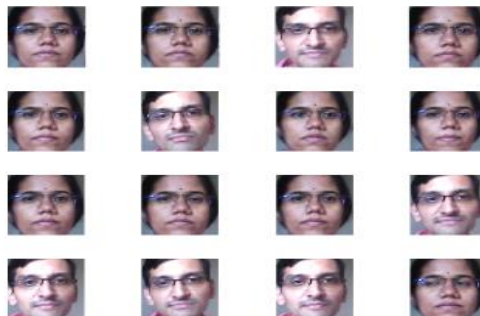
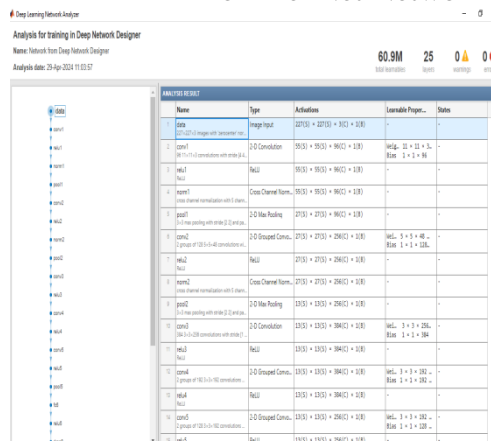


Fig. 7 Load data

3.4 Pre-trained CNN Model as Feature Extractor

Load a pretrained AlexNet network and the corresponding class names as shown in the fig 8. This requires the Deep Learning Toolbox™ Model for AlexNet Network support package.



Name	Type	Activation	Learnable Properties	Status
data	Image Input	(227/3) × (227/3) × (3/3) × (3/3)	-	-
conv1	2-D Convolution	(55/3) × (55/3) × (9/3) × (3/3)	W: 3 × 3 × 3 × 3, B: 1 × 1 × 3 × 3	-
relu1	ReLU	(55/3) × (55/3) × (9/3) × (3/3)	-	-
norm1	Conv Channel Normalization with 5 stats	(55/3) × (55/3) × (9/3) × (3/3)	-	-
pool1	2-D Max Pooling	(27/3) × (27/3) × (9/3) × (3/3)	W: 3 × 3 × 3 × 3, B: 1 × 1 × 3 × 3	-
conv2	2-D Grouped Convolution	(27/3) × (27/3) × (28/3) × (3/3)	W: 3 × 3 × 48 × 3, B: 1 × 1 × 3 × 3	-
relu2	ReLU	(27/3) × (27/3) × (28/3) × (3/3)	-	-
norm2	Conv Channel Normalization with 5 stats	(27/3) × (27/3) × (28/3) × (3/3)	-	-
pool2	2-D Max Pooling	(13/3) × (13/3) × (28/3) × (3/3)	W: 3 × 3 × 3 × 3, B: 1 × 1 × 3 × 3	-
conv3	2-D Convolution	(13/3) × (13/3) × (58/3) × (3/3)	W: 3 × 3 × 256 × 3, B: 1 × 1 × 3 × 3	-
relu3	ReLU	(13/3) × (13/3) × (58/3) × (3/3)	-	-
conv4	2-D Grouped Convolution	(13/3) × (13/3) × (58/3) × (3/3)	W: 3 × 3 × 192 × 3, B: 1 × 1 × 3 × 3	-
relu4	ReLU	(13/3) × (13/3) × (58/3) × (3/3)	-	-
conv5	2-D Grouped Convolution	(13/3) × (13/3) × (58/3) × (3/3)	W: 3 × 3 × 192 × 3, B: 1 × 1 × 3 × 3	-
relu5	ReLU	(13/3) × (13/3) × (58/3) × (3/3)	-	-

Fig 8.Details of AlexNet Layer in MATLAB Platform

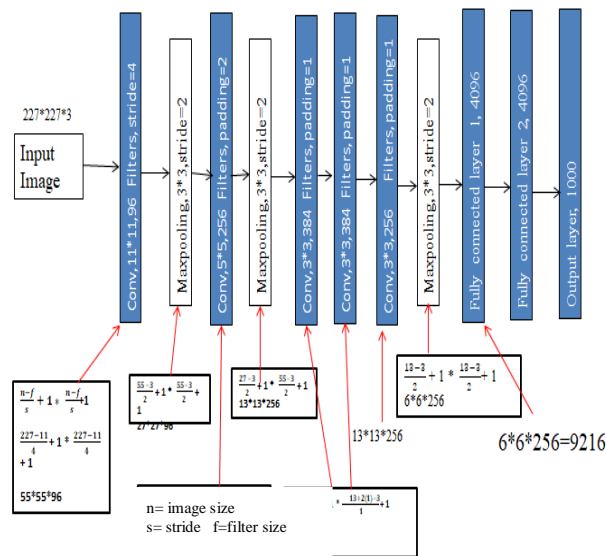


Fig 9 AlexNet Layer Structure (CNN) and Calculation of each layer

AlexNet has an image input size of 227x227x3 images with 'zero-center' normalization.

3.4.1 Convolutional layer 1

The first convolution layer filters the 227x227x3 image input image with 96 kernels of size 11x11x3 with a stride of 4 pixels and followed by ReLU non-linear activation, cross channel normalization with five channel per element, and 3x3 max pooling with the stride of 2 and zero padding layer as shown in the equation number 2. In equation number (2) n= image size, s=stride, f=filter size

$$\frac{n-f}{s} + 1 * \frac{n-f}{s} + 1 \quad (2)$$

$$\frac{227-11}{4} + 1 * \frac{227-11}{4} + 1$$

$$55 * 55 * 96$$

3.4.2 Convolutional layer 2

For the second convolution layer, 256 feature kernels with a size of 5x5x48 filters 27x27x96 feature image and carry out further feature extraction. Same as the first convolution layer, second convolution layer also followed by ReLU nonlinearity activation, cross channel normalization with 5 channels per element, and a 3x3 max pooling with the stride of 2 and zero padding and output of 13x13x256 image calculated as shown in equation number (3)

$$\frac{n+2p-f}{s} + 1 * \frac{n+2p-f}{s} + 1 \quad (3)$$

$$\frac{27+2(2)-5}{1} + 1 * \frac{27+2(2)-5}{1} + 1$$

$$27 * 27 * 256$$

3.4.3 Convolutional layer 3 and 4 Third and fourth convolution layers followed by ReLU non linearity activation given an output image of 13x13x384 as shown in the equation number 4.

$$\frac{13+2(1)-3}{1} + 1 * \frac{13+2(1)-3}{1} + 1 \quad (4)$$

$$13 * 13 * 384$$

3.4.4 Convolutional layer 5

Fifth convolution layer (conv5) filters 13x13x384 image by 256 feature kernels with 3x3x192 kernels size with padding of 1 and get an output of 13x13x256 image. As shown in the equation number (5)

$$\frac{13-3}{2} + 1 * \frac{13-3}{2} + 1 \quad (5)$$

$$6 * 6 * 256$$

3.4.5 Fully connected layer (FC6, FC7, FC8)

Then, it followed by a ReLU activation and a 3x3 kernel max pooling with the stride of 2 to 6x6x256 image.

$$\frac{13-3}{2} + 1 * \frac{13-3}{2} + 1 \quad (6)$$

6*6*256

Furthermore, fully connected layer 6(FC6) and fully connected layer 7(FC7) is followed by a ReLU activation and a 50% dropout to overcome over-fitting. A total of 4096 of 6x6x256 kernels perform convolution operation on the input data and output the operation result through 4096 neurons as shown in equation number (6). 4096 neurons from both 'FC6' and 'FC7' are fully connected. Lastly, fully connected layer 8 (FC8) which has 1000 neurons are connected to the input which consists of 4096 neurons from FC7 and softmax activation is used for classification. The detail of AlexNet layers in the MATLAB platform is shown in Fig 9. [19]

3.5 Train Network

Train network: although the photos in the image data stores vary in size, the network needs input images that are 227 by 227 by 3. Using an augmented image data store to specify further augmentation processes to be applied to the training photos and to automatically resize the images. The training images are randomly translated up to 30 pixels both horizontally and vertically, and they are randomly flipped along the vertical axis. By doing additional data augmentation, data augmentation helps avoid overfitting and the network's ability to automatically resize the validation photos by remembering every feature of the training images. employ an augmented picture data storage and provide the training options for transfer learning without specifying any additional pre-processing steps. For transfer learning, keep the features from the early layers of the pretrained network (the transferred layer weights). To slow down learning in the transferred layers, set the initial learning rate to a small value. In the previous step, you increased the learning rate factors for the fully connected layer to speed up learning in the new final layers. This combination of learning rate settings results in fast learning only in the new layers and slower learning in the other layers. When performing transfer learning, you do not need to train for as many epochs. An epoch is a full training cycle on the entire training data set. Specify the mini-batch size and validation data. When training a neural network using the `trainnet` function for classification, the software validates the network at each validation frequency iteration. By default, the `trainnet` function uses a GPU if one is available; to train on a GPU, one needs a parallel computing toolbox license and a supported GPU device; for more information on supported devices If not, the `trainnet` function uses the CPU to determine the execution environment; use the execution environment training option. See GPU computing requirements in the Parallel Computing Toolbox. separate the data into sets for training and validation. As can be seen in Figures 10 and 11, respectively, 70 of the photos are for training and 30 are for validation. Each label divides the image data store into two new data stores.

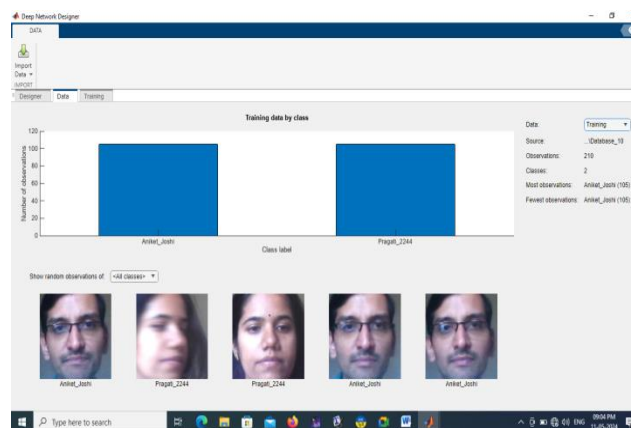


Fig 10 Classify Train Data Set of Images

3.6 Classify Validation Images

Before the model is finally put to the test, adjustments and improvements can be made to its parameters or hyperparameters based on the information provided by a validation dataset. Sort the photos for validation. Use the `minibatchpredict` function to generate predictions based on several observations. Use the `score2label` function to translate the prediction scores into labels. If a GPU is available, the `minibatchpredict` function utilizes it automatically. A Parallel Computing Toolbox™ license and a compatible GPU device are needed in order to use a GPU.

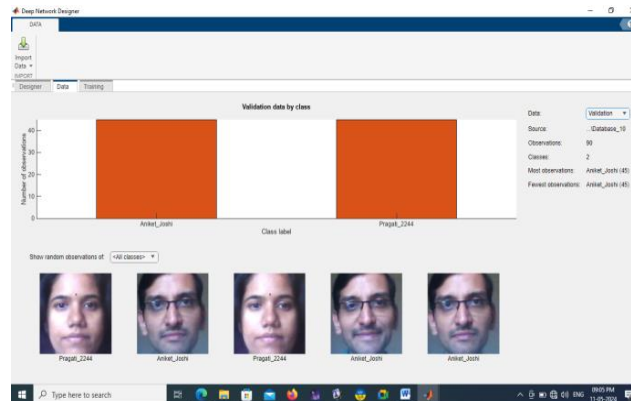


Fig 11 Classify Validation Data Set of Images

D Stochastic Gradient Descent

Stochastic gradient descent (SGD) is machine learning's workhorse optimization algorithm. SGD trains several orders of magnitude faster than methods such as batch gradient descent, with no loss of model accuracy. This is due to how we calculate the gradient for a single input training example (or mini-batch of training examples). The computed gradient is a "noisy" approximation of the true gradient yet allows SGD to converge faster. The strengths of SGD are easy implementation and the quick processing of large datasets. You can adjust SGD by adapting the learning rate. SGD is also a popular algorithm for training neural networks due to its robustness in the face of noisy updates. That is, it helps you build models that generalize well.[7] In gradient descent we'd calculate the overall loss across all of the training examples before calculating the gradient and updating the parameter vector. In SGD, we compute the gradient and parameter vector update after every training sample. This has been shown to speed up learning and also parallelizes. SGD is an approximation of "full batch" gradient descent. Mini-batch training and SGD Another variant of SGD is to use more than a single training example to compute the gradient but less than the full training dataset. This variant is referred to as the mini-batch size of training with SGD and has been shown to be more performing than using only single training instances. Applying mini-batch to stochastic gradient descent has also shown to lead to smoother convergence because the gradient is computed at each step it uses more training examples to compute the gradient. As the mini-batch size increases the gradient computed is closer to the "true" gradient of the entire training set. This also gives us the advantage of better computational efficiency. If our mini-batch size is too small (e.g., 1 training record), we're not using hardware as effectively as we could, especially for situations such as GPUs. Conversely, making the mini-batch size larger (beyond a point) can be inefficient, as well, because we can produce the same gradient with less computational effort (in some cases) with regular gradient descent [7].

E Algorithm:

Data Collection:

1. Initialize the system by clearing the console, workspace, and closing all figures.
2. Suppress warnings.
3. Initialize a webcam object.
4. Initialize a face detector using the `CascadeObjectDetector`.

5. Set the maximum number of images to collect (**c**) and initialize a variable **temp** to keep track of the current number of collected images.
6. Start an infinite loop.
 - Capture a snapshot from the webcam.
 - Detect faces in the captured image.
 - Check if any faces are detected.
 - If faces are detected and the number of collected images is greater than or equal to **c**, exit the loop.
 - If faces are detected and the number of collected images is less than **c**, extract the face region, resize it, save it to a file, and display it.
 - If no faces are detected, display the original image.
7. End of the loop.

Training Model:

1. Load the pre-trained AlexNet model.
2. Modify the last two layers of the model to match the number of classes (2 classes: face and non-face).
3. Create an imageDatastore to store the training images, specifying the folder structure and label source.
4. Define training options including the optimizer, initial learning rate, maximum epochs, and mini-batch size.
5. Train the network using the training data and specified options.
6. Save the trained model.

Testing Model:

1. Initialize a webcam object.
2. Load the trained model.
3. Initialize a face detector using the CascadeObjectDetector.
4. Start an infinite loop.
 - Capture a snapshot from the webcam.
 - Detect faces in the captured image.
 - Check if any faces are detected.
 - If faces are detected, extract the face region, resize it, classify it using the trained model, display the image with the predicted label.
 - If no faces are detected, display the original image with a message indicating no face is detected.
5. End of the loop.

F Flowchart

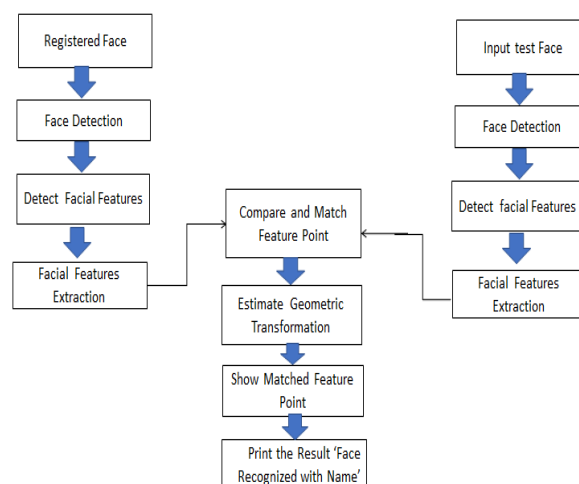


Fig 12 Flowchart of Face Detection and Recognition model

III Result and Discussion

In this session, all the results are presented in graph, table and images. All results are discussed in detail. we have done different experiments to evaluate performance approach such as fine tuning of hyper parameter to get better accuracy.

A Setting

Hyper Parameter Tuning to avoid over fitting. Table 2 describes parameters used in training of data.

Hyper-parameter	Selected value
Learning Rate	0.0001
Epochs	20
Mini-batch size	64
Executive Environment	CPU

Table 2: Hyper - parameter Settings.

B Training Progress Graph

The performance of CNN model can be evaluated and observed from the training progress graph of Accuracy vs. Loss as shown in the fig.13

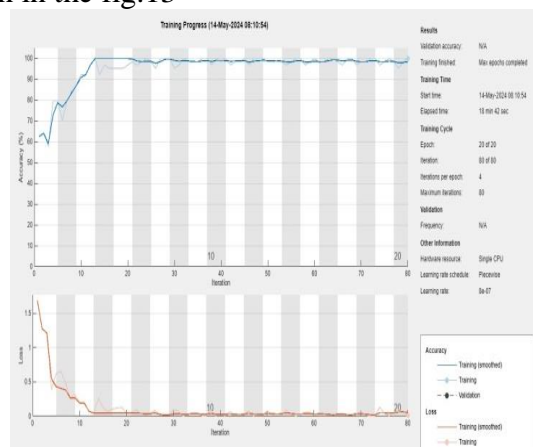


Fig 13 Training Progress Graph for Accuracy and Loss

Above Training progress plot evaluate Accuracy and Loss and observed that trained model achieved 100% accuracy at 3rd epoch and simultaneously reduced loss up to 0.0065.

C Face Detection and Recognition from Real Time Image

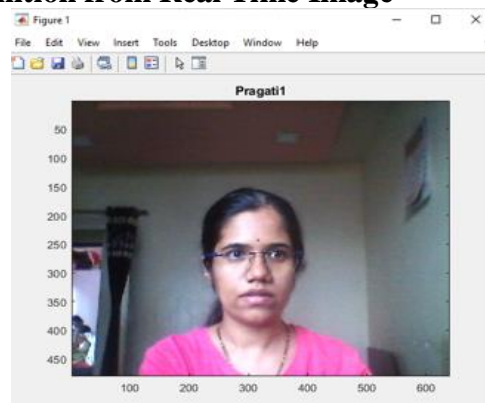


Fig 14 Result of Database 1 ‘Pragati’

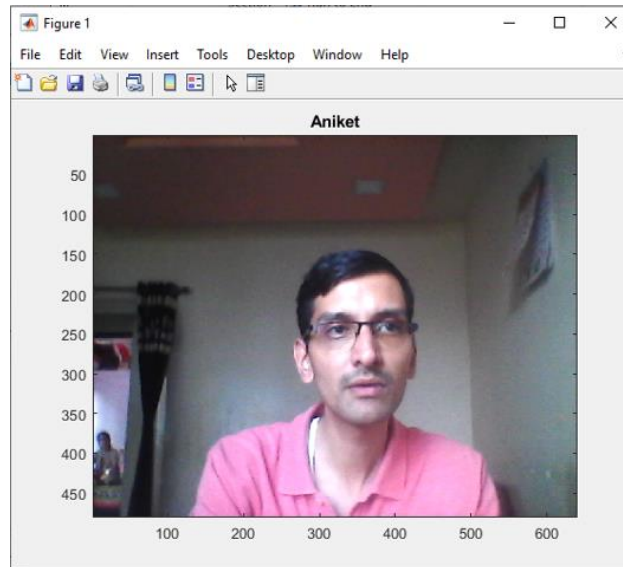


Fig 15 Result of Database 2 ‘Aniket’

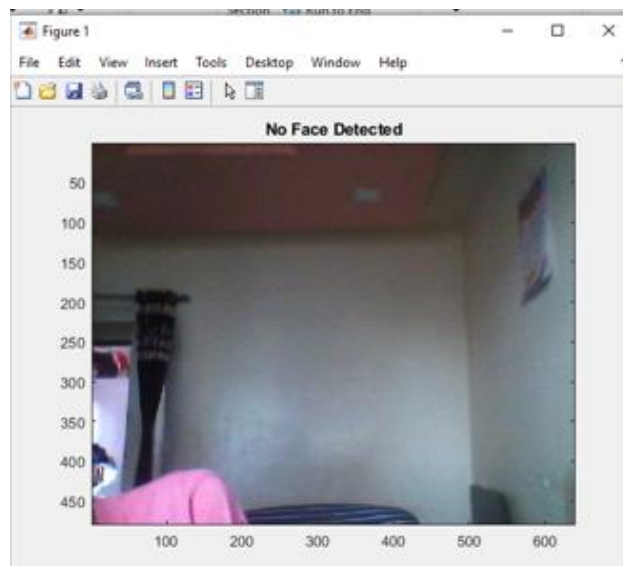


Fig 16 Result labelled with ‘No Face Detected’

Above mentioned Fig 14 and Fig 15 shows result for human faces with 100% accuracy also it labelled Fig 16 as ‘No Face Detected’ if image does not have any human face within frame.

References

- [1] Ms. Gayatri Ramrakhiani, Face Detection using Viola Jones Algorithm and Neural Networks 978-1-5386-5257-2/18/\$31.00 ©2018 IEEE
- [2] PDF By-Dr. S R Prasad Assistant Professor, Department of E&TC Engineering, DKTE Society’s Textile and Engineering Institute, Ichalkaranji, Dist.- Kolhapur
- [3] Gonzales and Woods, “Digital Image Processing”, Pearson Education, India, Third Edition.
- [4] P. Viola and M. Jones, “Robust real-time face detection”, International Journal of Computer Vision, 57(2), 2004, pp 137–154
- [5] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

- [6] Monali Chaudhari, Shantasondur, Gauresh Vanjare. "A review on Face Detection and study of Viola Jones method", IJCTT, 2015.
- [7] [DeepLearningPractitionersApproach.pdf](#)
- [8] Chandrappa D N, M Ravishankar, D R Ramesh Babu, "Face Detection in Color Images using Skin Color Model Algorithm based on Skin Color Information", Electronics Computer Technology (ICECT), 2011 3rd International Conference.
- [9] S.N. Sivanandam and S.N. Deepa, "Principles of Soft Computing", Wiley, India. Second Edition.
- [10] Transfer Learning Using AlexNet from the below link: <https://www.mathworks.com/help/deeple...>
- [11] ImageNet Classification with Deep Convolutional Neural Networks Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton
- [12] Lienhart R., Kuranov A., and V. Pisarevsky "Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection." Proceedings of the 25th DAGM Symposium on Pattern Recognition. Magdeburg, Germany, 2003.
- [13] Ojala Timo, Pietikäinen Matti, and Mäenpää Topi, "Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns" . In IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002. Volume 24, Issue 7, pp. 971-987.
- [14] Kruppa H., Castrillon-Santana M., and B. Schiele. "Fast and Robust Face Finding via Local Context" . Proceedings of the Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2003, pp. 157–164.
- [15] astrillón Marco, Déniz Oscar, Guerra Cayetano, and Hernández Mario, " ENCARA2: Real-time detection of multiple faces at different resolutions in video streams" . In Journal of Visual Communication and Image Representation, 2007 (18) 2: pp. 130-140.
- [16] Yu Shiqi " Eye Detection." Shiqi Yu's Homepage. <http://yushiqi.cn/research/eyedetection>.
- [17] D. S. Trigueros, Li Meng, and Margeret Hartnett, "(PDF) Face Recognition: From Traditional to Deep Learning Methods," ResearchGate, 2018.
- [18] M. D. Kelly, Visual identification of people by computer. 1971.
- [19] Transfer learning using AlexNet Convolutional Neural Network for Face Recognition Ridza Azri Bin Ramlee, Yvonne Yap, Siva Kumar Subramaniam, Mohamad Harris Misran, Asem Khmag