Indian Institute of Space Science and Technology

Machine Learning For Signal Processing
Course Project Presentation on

Face Detection and Recognition Based on Eigenface Method

Completed Under the guidance of

Dr. Deepak Mishra,

Associate Professor and Head, Department of Avionics, IIST

Parthasarthi Samanta (SC17B106)

Kiran, L. (SC17B150)

Outline:

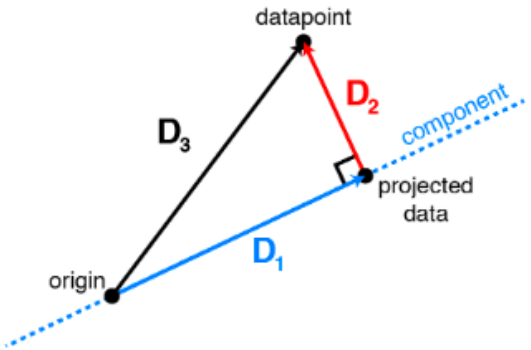
Introduction

Principal Component Analysis (PCA)

PCA can be viewed in the following ways:

- One of the more-useful methods from applied linear algebra
- Non-parametric way of extracting meaningful information from confusing data sets.
- Uncovers hidden, low-dimensional structures that underlie your data.

Orthogonal Projection



$$D_3^2 = D_1^2 + D_2^2$$

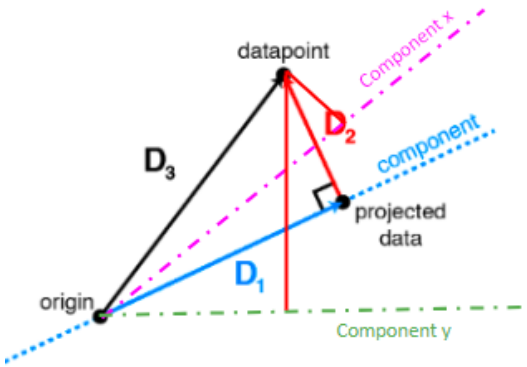
initial variance = remaining variance + lost variance

$$\|a_i\|^2 = \|w_i c\|^2 + \|a_i - w_i c\|^2$$

this is constant — maximize this **or** minimize this

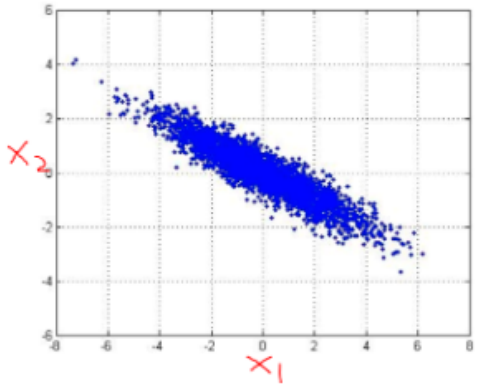
- The projection of Ai vector onto the principal components relates the remaining variance to the squared residual by the Pythagorean theorem..
- Choosing the components to maximize variance is the same as choosing them to minimize the squared residuals..

Orthogonal Projection



- Projection on component x has smaller lost variance than component and component y.
- Projection on component y has higher lost variance than component and component x.

Computing Principal Components

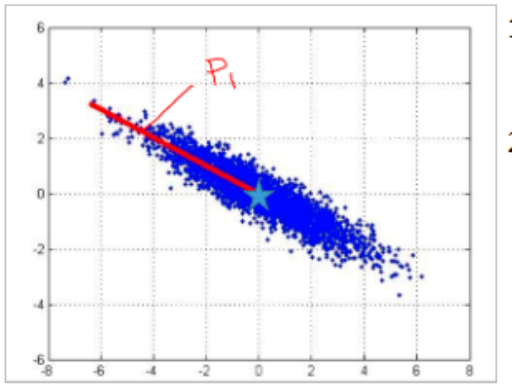


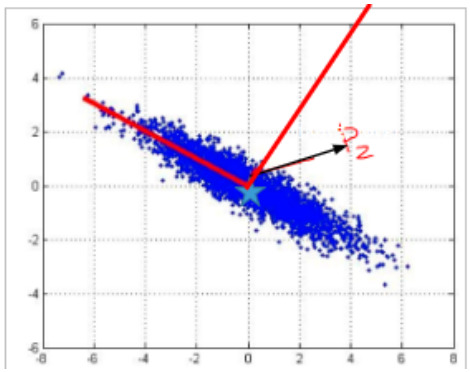
PCA can be obtained in following ways-

- Start with the centroid (Xavg, Yavg)
- Determine the direction along which the sum of projection error /residual is minimised. We call this 'first' principal component.
- The 'second' principal component is the direction which is orthogonal to the first one.

This is a sequential algorithm.

PCA



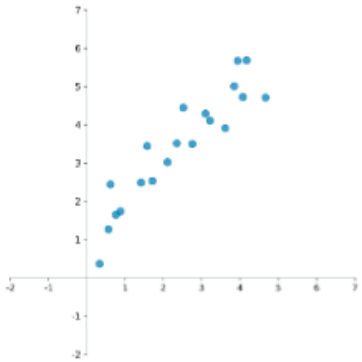


PCA

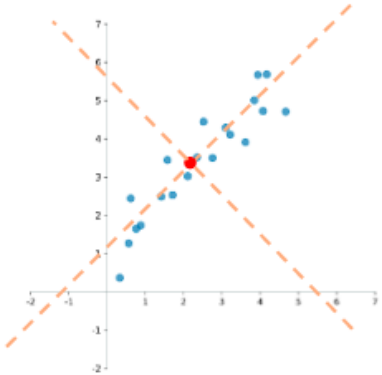
There are two other more elegant algorithms to compute PCA:

- Eigen Decomposition of Sample Covariance Matrix
- Singular Value Decomposition of Data Matrix (below section)

PCA Using Scikit-Learn

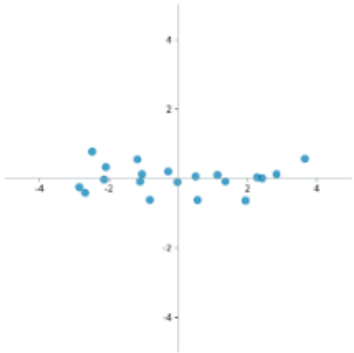


```
from sklearn.decomposition import PCA

data = data = np.array([
 [1.41205819, 2.48438943],
 [0.87435703, 1.73657007],
 [3.84516794, 5.00362514],
 …
 [2.51401195, 4.44218097],
 [4.66614767, 4.70587771],
 [0.75369264, 1.64649525],
 [3.93658659, 5.66619787]
])
```



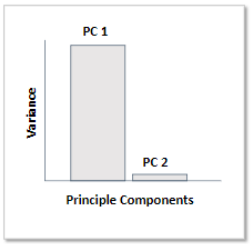
```
from sklearn.decomposition import PCA

data = data = np.array([
 [1.41205819, 2.48438943],
 [0.87435703, 1.73657007],
 [3.84516794, 5.00362514],
 …
 [2.51401195, 4.44218097],
 [4.66614767, 4.70587771],
 [0.75369264, 1.64649525],
 [3.93658659, 5.66619787]
])

pca = PCA(n_components = 2)
pca.fit(data)
```

PCA Using Scikit-Learn

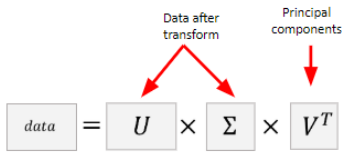


```
from sklearn.decomposition import PCA

data = data = np.array([
 [1.41205819, 2.48438943],
 [0.87435703, 1.73657007],
 [3.84516794, 5.00362514],
 …
 [2.51401195, 4.44218097],
 [4.66614767, 4.70587771],
 [0.75369264, 1.64649525],
 [3.93658659, 5.66619787]
])

pca = PCA(n_components = 2)
pca.fit(data)

z = pca.transform(data)
```

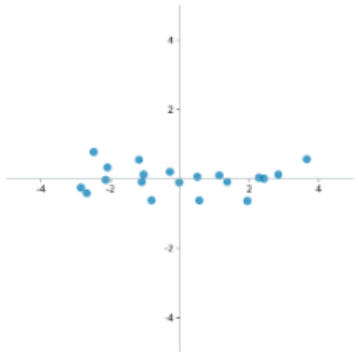
```
features = range(pca._components_)

plt.bar(features, pca.explained_variance_ )

plt.xticks(features)
plt.xlabel("Principal Components")
plt.ylabel("Variance")
```



PCA from Numpy SVD




```
import numpy as np

data = data = np.array([
 [1.41205819, 2.48438943],
 [0.87435703, 1.73657007],
 [3.84516794, 5.00362514],
 …
 [2.51401195, 4.44218097],
 [4.66614767, 4.70587771],
 [0.75369264, 1.64649525],
 [3.93658659, 5.66619787]
])

u,s,vt = np.linalg.svd(data)
```



```
import numpy as np

data = data = np.array([
 [1.41205819, 2.48438943],
 [0.87435703, 1.73657007],
 [3.84516794, 5.00362514],
 …
 [2.51401195, 4.44218097],
 [4.66614767, 4.70587771],
 [0.75369264, 1.64649525],
 [3.93658659, 5.66619787]
])

u,s,vt = np.linalg.svd(data)

z = np.dot(u, s)
```

Objective

The main objective of this project is to design a system able to take in image(s), locate and identify people's faces using Eigenface method. Thus, the two broad sub-objectives are:

1. Face Detection
2. Face Recognition

Problem Statement

One of the main issues with face detection using conventional template matching methods is expensive in terms of computation, time and storage.

Thus, using Principal Component Analysis, a machine learning based system is to be designed that can achieve the aforementioned objectives, i.e., detect and identify people based on their frontal facial features.

Literature Survey

Real life situations where face detection and recognition system finds its application

- Criminology (convicts' database)
- Facebook (auto-tagging feature)
- CCTV based Patrolling (improve efficiency and serve as proof)
- ATMs (additional security)

Literature Review

Previous works related to face detection and recognition:

1. "Face Recognition System With Face Detection": Final Year Project by a team of 4 students from Jawaharlal Nehru Technological University, Kakinada, completed under the guidance of Ms. SK. AYESHA, M.Tech, Assistant Professor of E.C.E dept during the academic year 2013-17.
    1. Implemented in MATLAB
    2. Face detection through face localization - followed by mouth, nose and eyes and verifying eyes-mouth triangle.
    3. Fully automated face detection implemented using a deformable template algorithm relying on image invariants of human faces.
    4. Face recognition based on PCA with 30 test subjects.

Literature Review

2. A Face recognition system was built by Yang Li and Sangwhan Cha, PhD. Assistant Professor of Computer Science, in which ORI dataset of 400 frontal face images (of 40 subjects, with 10 faces per person) was used.
    1. Siamese network designed for measuring similarity between two subject's faces.
    2. Model training on Nvidia GPU using PyTorch.
    3. No face detection

Literature Review

3. Open Face: Implemented in Python and Torch, the workflow of the system includes following steps:
   1. Detect faces with a pre-trained models from dlib or OpenCV,
   2. transform the face for the neural network using OpenCV's affine transformation,
   3. crop out the face from the image,
   4. use a deep neural network for training and
   5. use the trained model classifier for clustering, similarity detection and classification.

Data

In this project, 3 different datasets have been used, as briefed below:

1. **Washington Dataset:**
   1. colored images of 24 subjects
   2. 2 images per person - smiling and non-smiling
   3. provide separate test set
2. **Fetch Olivetti Faces Dataset:**
   1. Black and white images of 40 Subjects
   2. 10 faces per person
   3. pre-processed images: cropped, face-aligned, histogram equalized.
   4. provide no separate test set
3. **Labeled Faces in the Wild Home (LFW) Dataset**
   1. 13233 images
   2. 5749 people with two or more images of 1680 people

   Only a subset of 93 subjects out of 5749 considered with 3 or 4 faces per person.

Training Pipeline

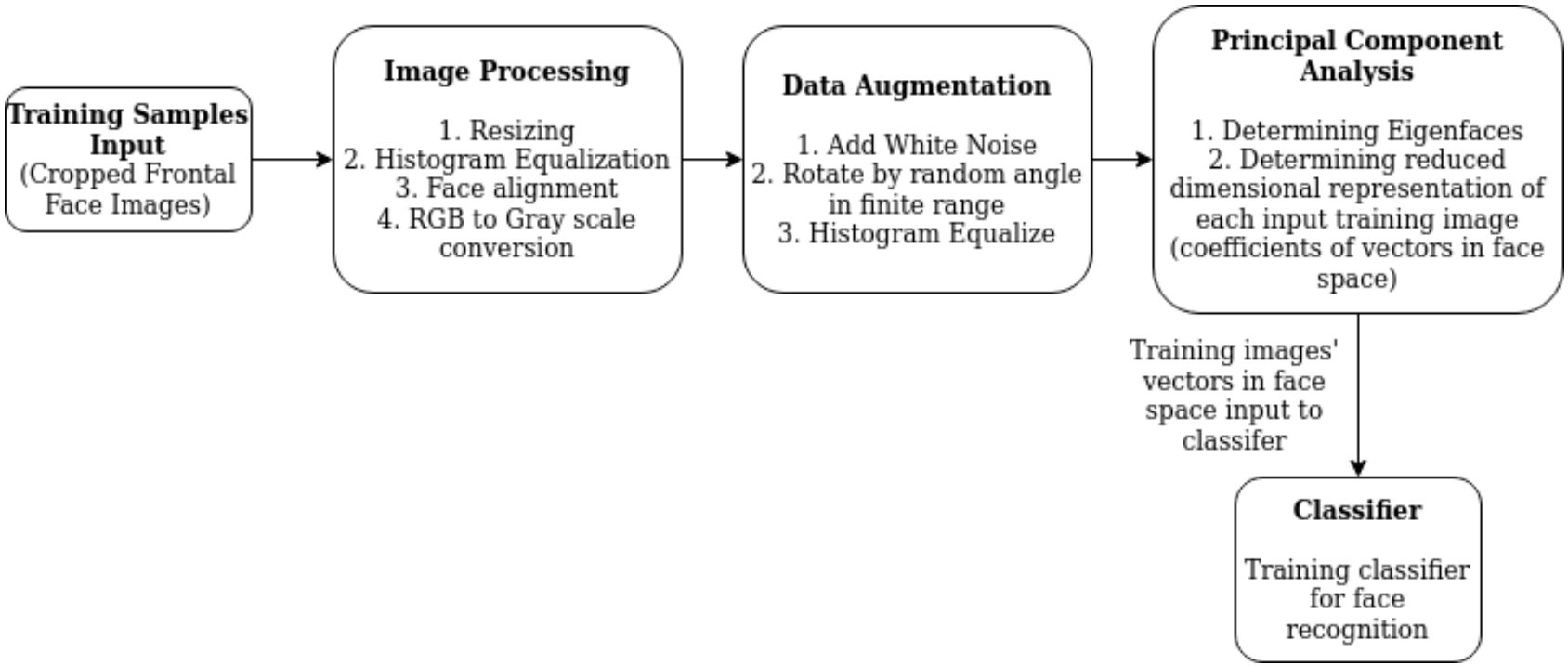


Figure: Training Pipeline

Testing Pipeline

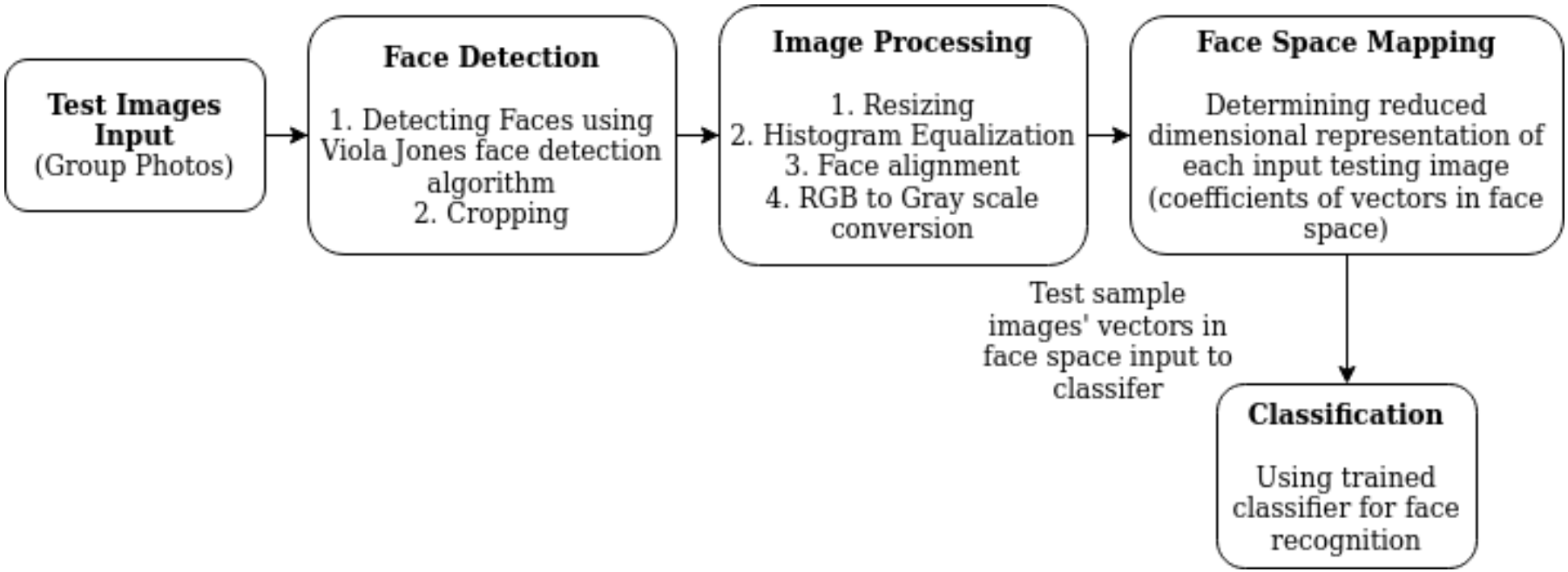


Figure: Testing Pipeline

Principal Component Analysis

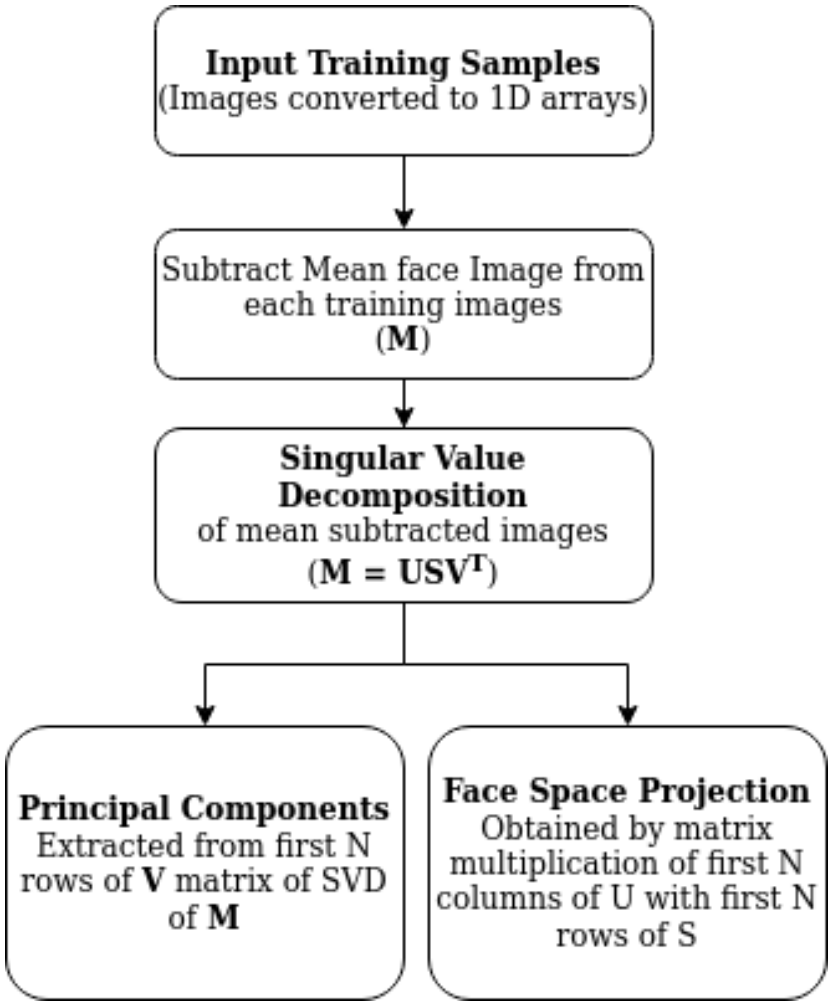


Figure: Principal Component Analysis Block Diagram

Design

For the Classifier Design we have used Different Types of Classifiers-

- K-Nearest Neighbor
- Support Vector Machine
- Neural Network
- Ensemble Classifier

KNN-Basic Analogy

Simple Analogy..

- Tell me about your friends(*who your neighbors are*) and *I will tell you who you are*.

KNN-Different Names

- K-Nearest Neighbors
- Memory-based Reasoning
- Example-based Reasoning
- Instance-Based Learning
- Lazy learning

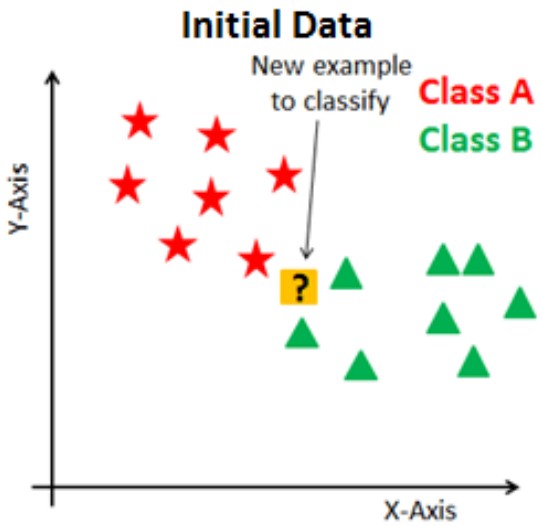
What is KNN

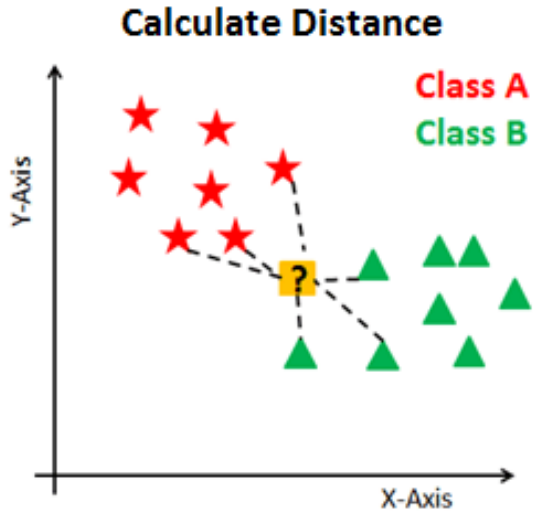
- KNN is a very simple but powerful algorithm used in pattern recognition
- It stores all available cases (training set) and classifies new classed based on similarity measure.
- A non-parametric lazy learning algorithm.

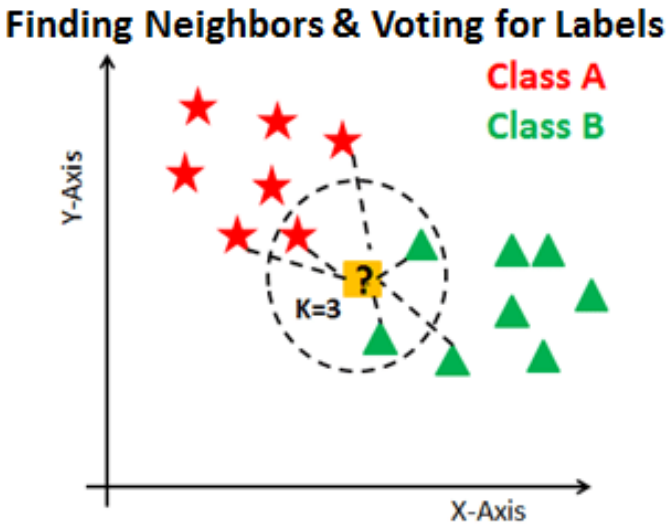
KNN: Classification Approach

- An object (a new instance) is classified based upon the voting by its K-nearest samples' classes.
- The Similarity measure is done using a distance functon.

KNN-Classification







Distance Measure Functions

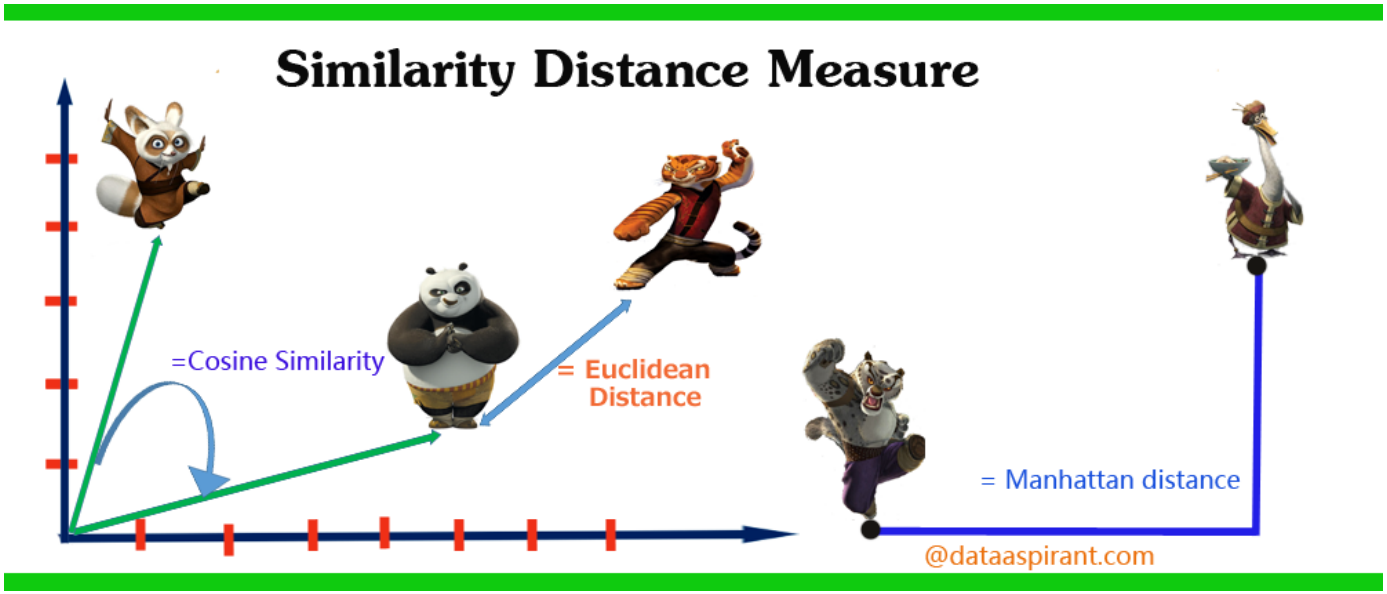
Distance functions

Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan $$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski $$\left(\sum_{i=1}^{k}\left(|x_i - y_i|\right)^q\right)^{1/q}$$

Similarity Distance Measurement



KNN Algorithm Explanation

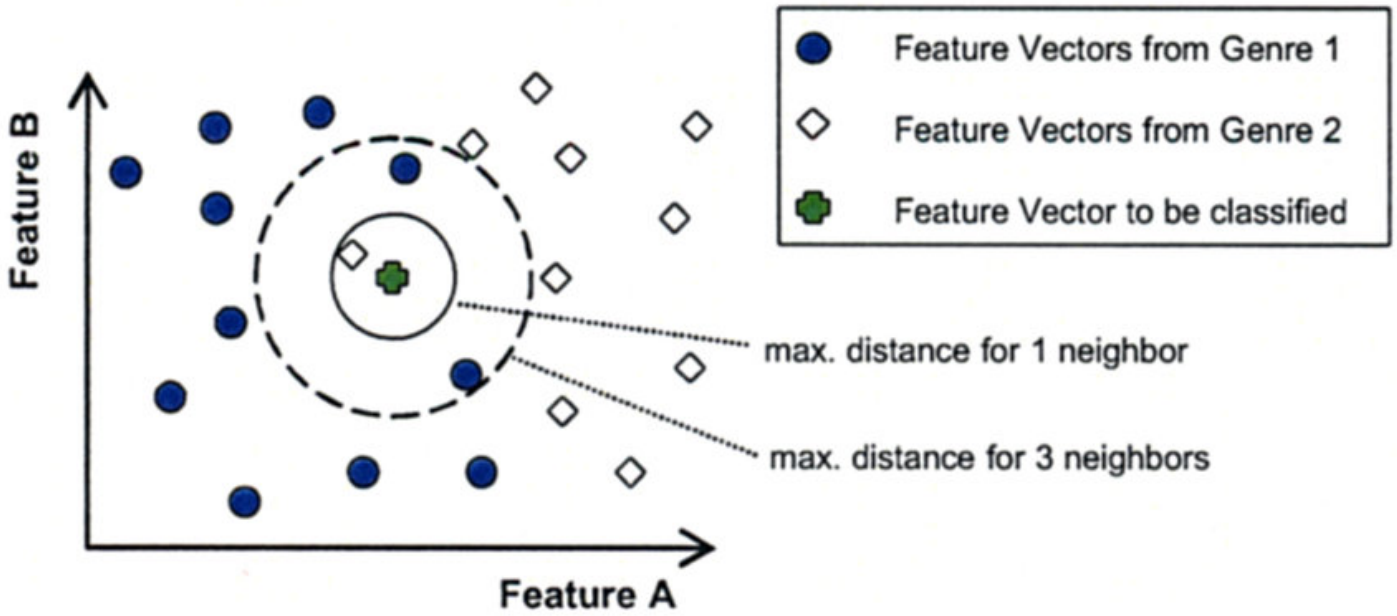
- All the instances correspond to the points in N-Dimensional feature Space.
- Each instance is represented with a set of numerical attributes.
- Each of the training data consists of a vector and a Class label attributed to the vector.
- Classification is done by comparing the feature vectors of a new sample with its nearest neighbors' vectors using the distance functions.
- Assign the new point to the most common and frequent class in its neighbors.

How to Choose K?

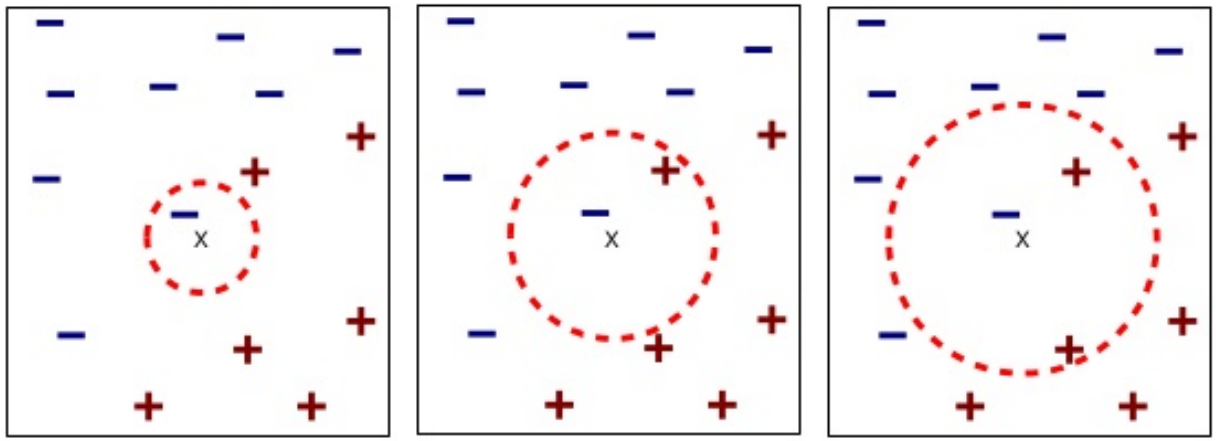
- If K is too small the classification may be sensitive to noise (for example let's assume there is only one training sample from a particular class in the middle of all other samples from all different classes, now this outlier can cause misclassification).
- If K is too large then there may be one cluster of samples from a particular class around a test sample which will lead to biased decisions.
- The rule of thumb is K $< \sqrt{N}$, where N is number of samples with K being odd to avoid tie.

KNN the choice of K

- As we explained earlier, 1 NN can be very sensitive to outlier as it can be seen from the figure below.



KNN Example



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

16

KNN Advantages and Disadvantages

Advantages

- Very simple and Intuitive.
- Can be applied data from any distribution.
- If the number of samples is large enough then a good classifier

Disadvantages

- Need large number of samples for better accuracy.
- Choice of K is trade-off between performance and speed.

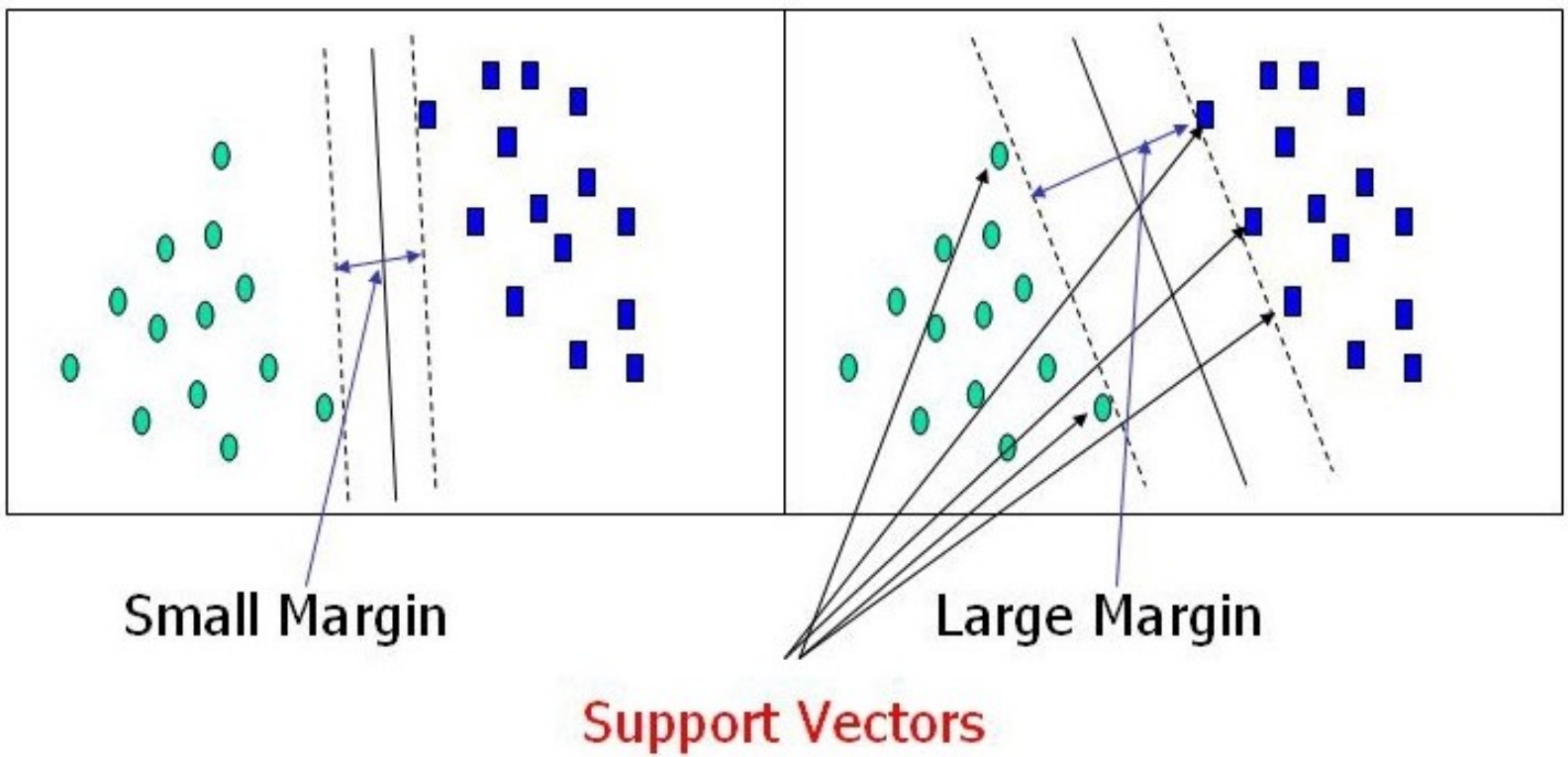
SVM-Support Vector Machine

Introduction

Support Vector Machines (SVMs) are a set of supervised learning methods which learn from the dataset and can be used for both regression and classification.

An SVM is a kind of large-margin classifier: it is a vector space based machine learning method where the goal is to find a decision boundary between two classes that is maximally far from any point in the training data.

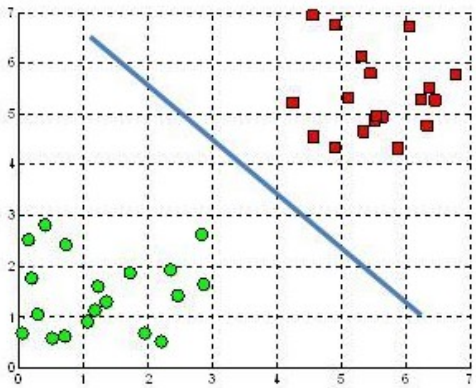
SVM-Basics



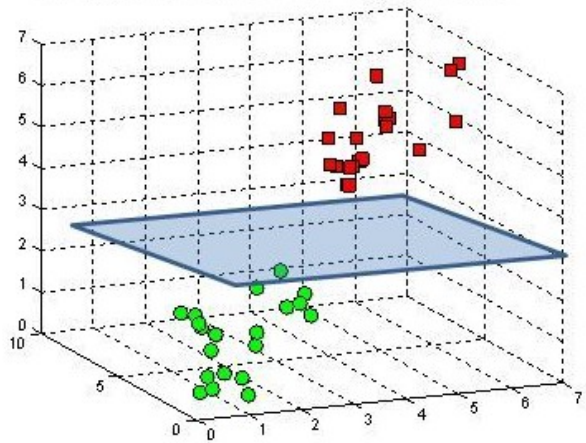
Support Vectors

The term Support Vectors refers to the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes using a hyperplane/ line.

A hyperplane in $\mathbb{R}^2$ is a line



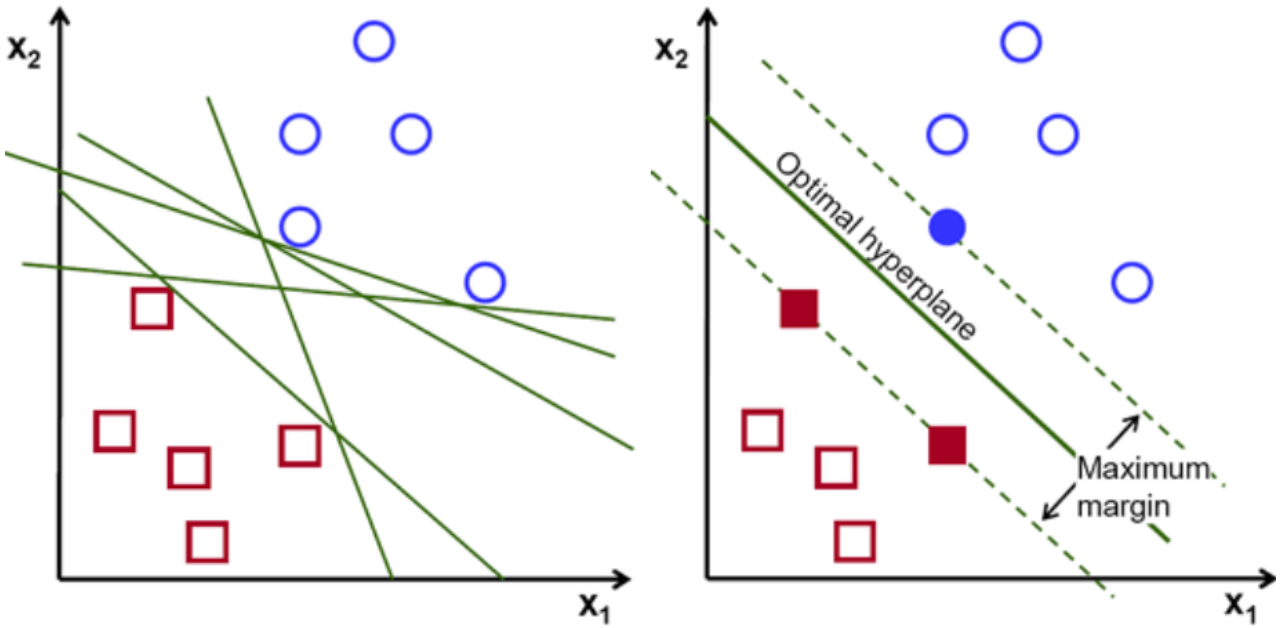
A hyperplane in $\mathbb{R}^3$ is a plane



A hyperplane in $\mathbb{R}^n$ is an n-1 dimensional subspace

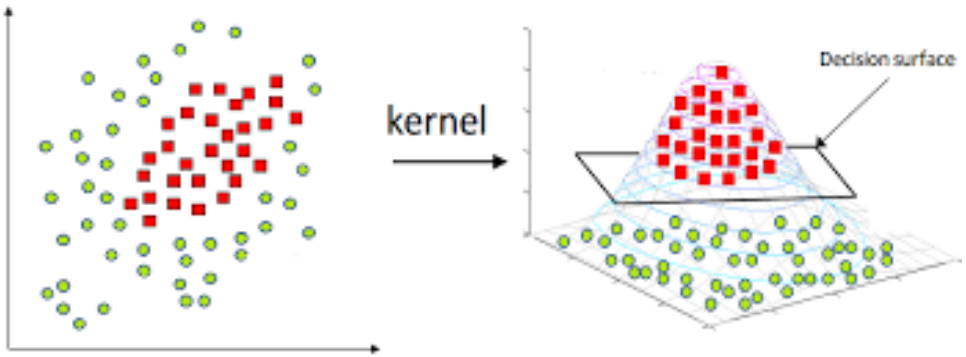
Working of SVM

- An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.
- New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.



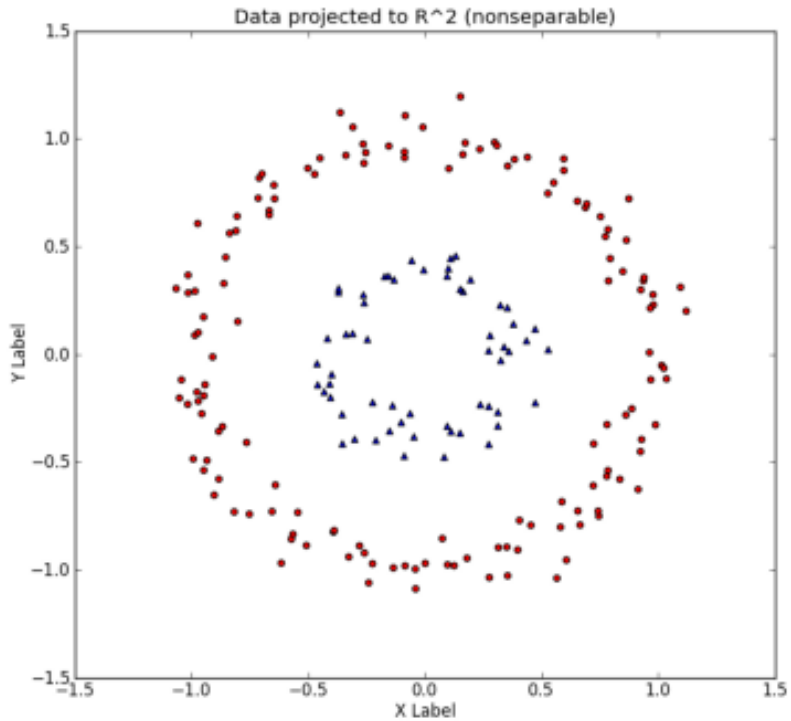
Non-linear Classification- Kernel Trick

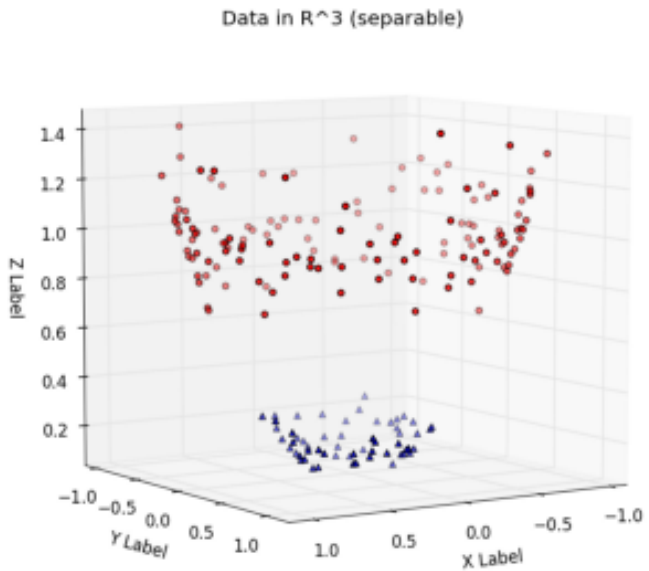
In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.



Kernel Method

- Kernel method is used by SVM to perform a non-linear classification. They take low dimensional input space and convert them into high dimensional input space.
- It converts non-separable classes into the separable one, it finds out a way to separate the data on the basis of the data labels defined by us.





SVM-Support Vector Machine

Advantages

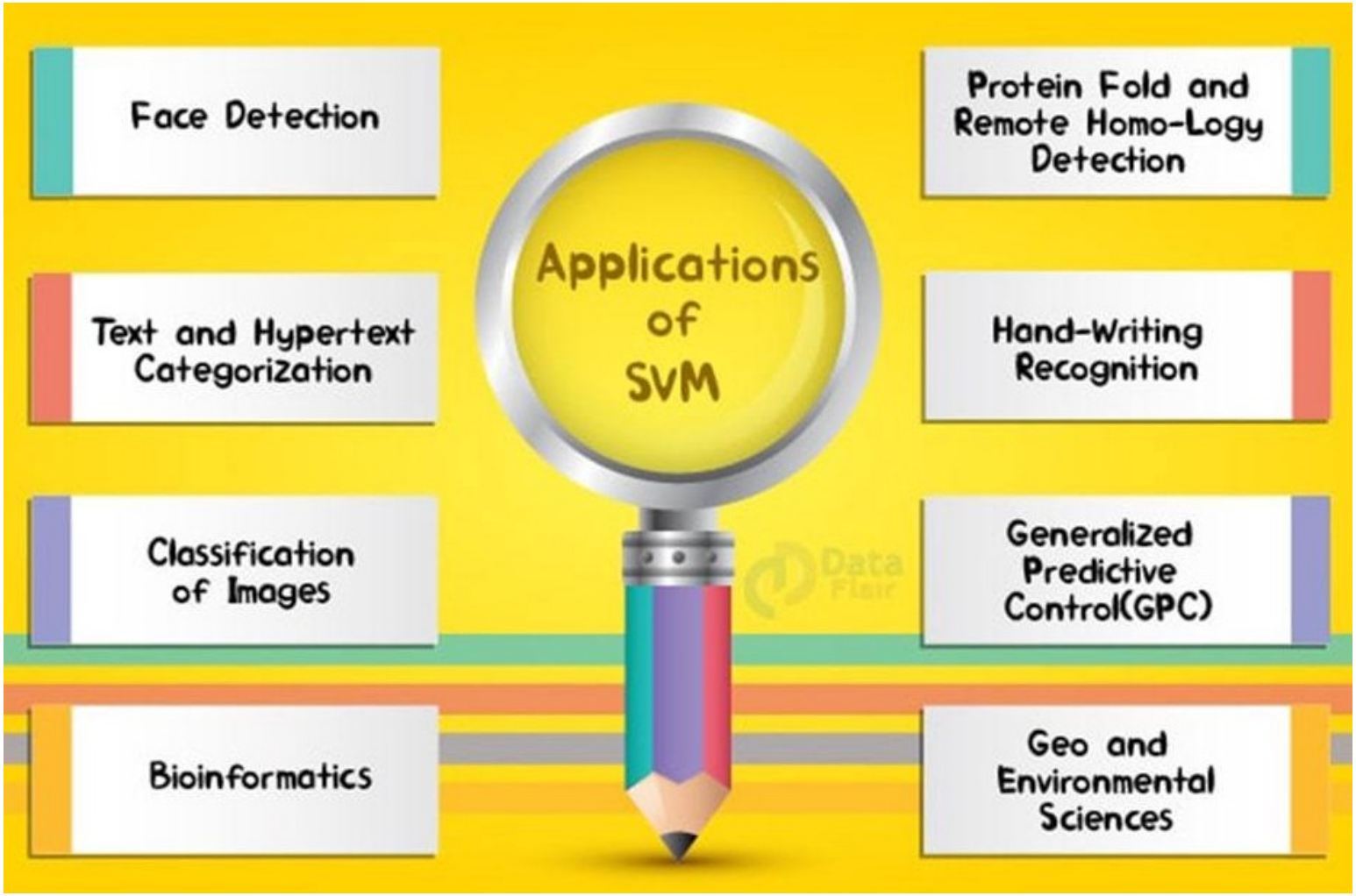
- They maximize the margin of the decision boundary using quadratic optimization techniques which find the optimal hyperplane.
- It has the ability to handle large feature spaces.
- SVM's are very good when we have no idea about our data.
- Works well with even unstructured and semi-structured data like text, Images and trees.
- The kernel trick is real strength of SVM. With an appropriate kernel function, we can solve any complex problem.
- It scales relatively well to high dimensional data.
- SVM models have generalization in practice, the risk of over-fitting is less in SVM.

SVM-Support Vector Machine

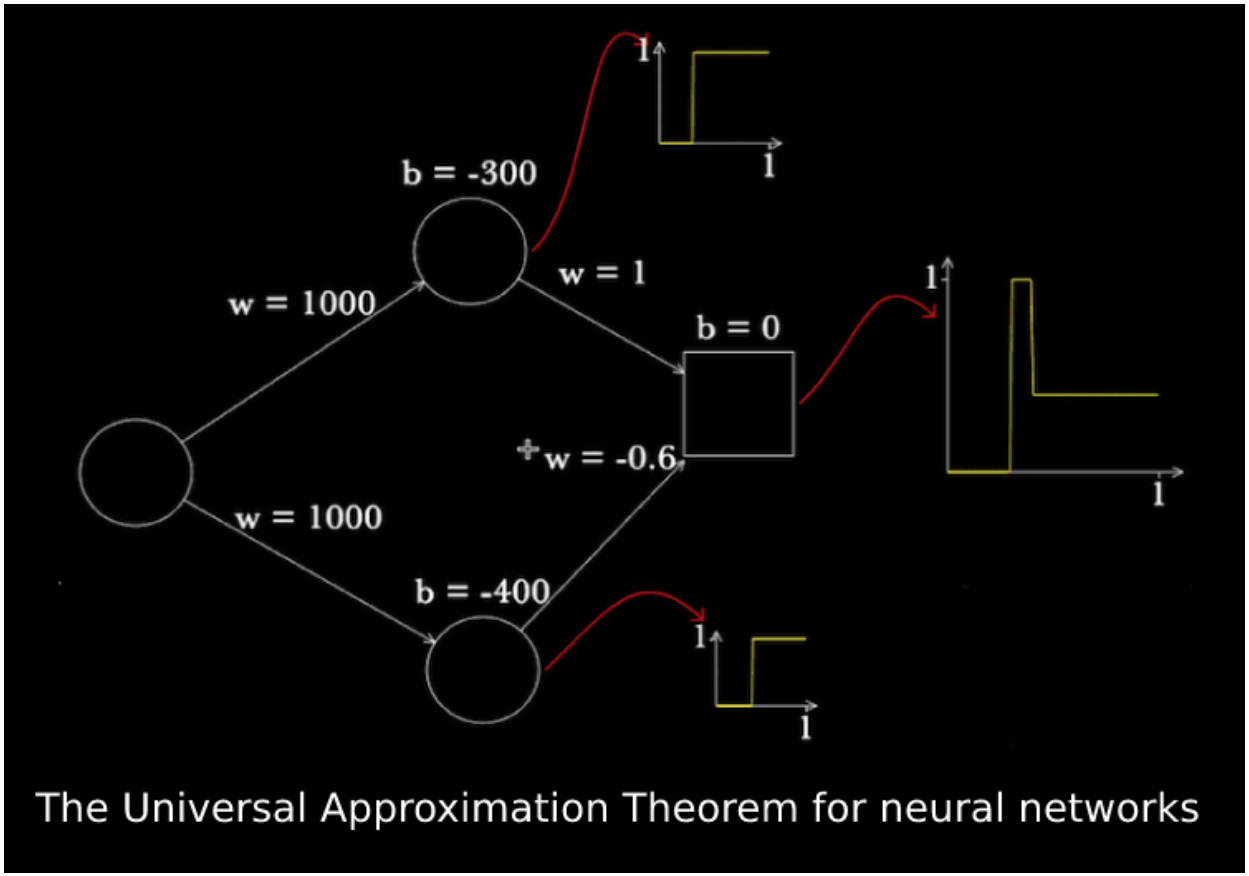
Disadvantages

- It is sensitive to noise.
- The extension of classification to more than two classes is problematic.
- Choosing a "good" kernel function is not easy.
- The SVM hyper parameters are Cost -C and gamma. It is not that easy to fine-tune these hyper-parameters. It is hard to visualize their impact.

Some applications of SVM



Universal Approximation Theorem



Universal Approximation Theorem

- The universal approximation theorem (UAT) states that a single hidden layer neural network with finite neurons under mild assumptions on the activation function can approximate continuous functions on compact subsets of R^n.
- It says nothing about algorithmic learnability of those parameters.
- One of the first versions of the UAT for sigmoid activation functions was proved by George Cybenko in 1989.
- Hornik showed in 1991 that it is not the specific choice of the activation function, but rather the choice of the multilayer feed forward architecture which makes neural networks potential universal approximators.
- The output units of the same are always assumed to be linear.

Feed Forward Neural Network

Basic Idea

- A neural network simply consists of neurons (also called nodes). These nodes are connected in some way.
- Then each neuron holds a number, and each connection holds a weight.
- These neurons are split between the input, hidden and output layer. In practice, there are many layers and there are no general best number of layers.

Input Layer

- The input data is just the dataset, where each observation is run through sequentially from $x = 1, ..., x = i$.
- Each neuron has some activation — a value between 0 and 1, where 1 is the maximum activation and 0 is the minimum activation.

Feed Forward Neural Network

Input Layer to Hidden Layer

- Each neuron has an activation a and each neuron that is connected to a new neuron has a weight w.
- Activations are typically a number within the range of 0 to 1, and the weight is a double.
- One could multiply activations by weights and get a single neuron in the next layer, from the first weights and activations $w_1 a_1$ all the way to $w_n a_n$:
  $w_1 a_1 + w_2 a_2 + ... + w_n a_n =$ *new neuron*.

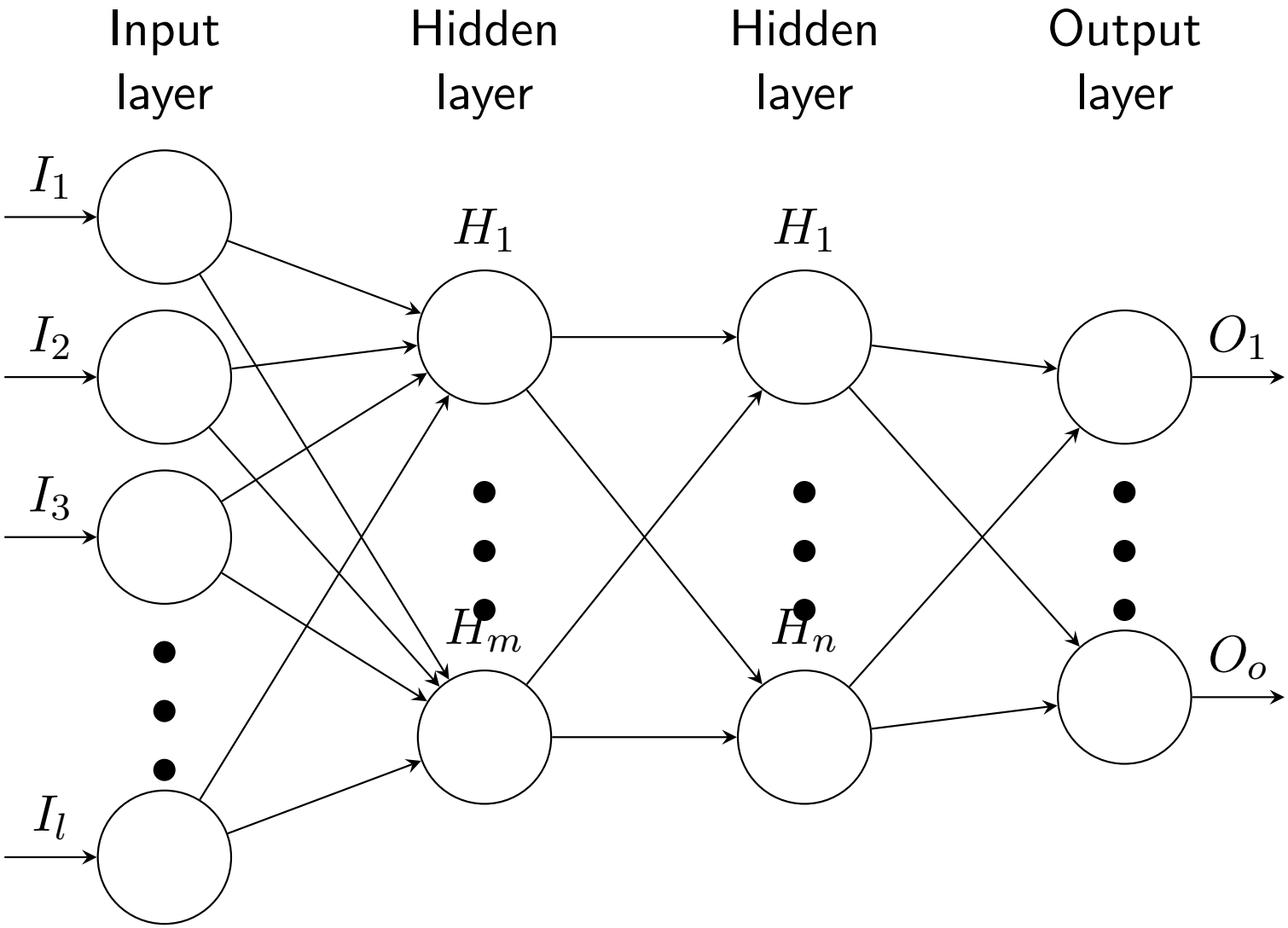
The procedure is the same moving forward in the network of neurons, hence the name feed forward neural network, and finally we reach the output layer.

Feed Forward Neural Network

Hidden Layer to Output Layer

- The neuron outputs from the final hidden layer is given to the Output layer, which contains neurons among which one activation will be higher than all others corresponding to a particular class.
- Sometimes it can be related to a probability, where each neuron in the output layer contains some activation value(some small positive value for all neurons except the one corresponding to the right class which will have activation number close to 1.)

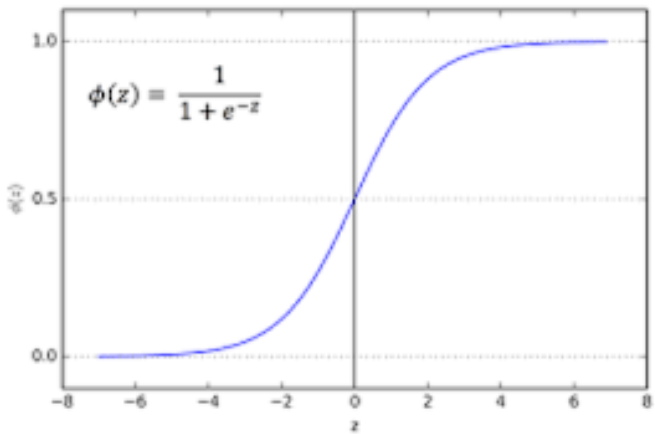
Feed Forward Neural Network



Activation Functions

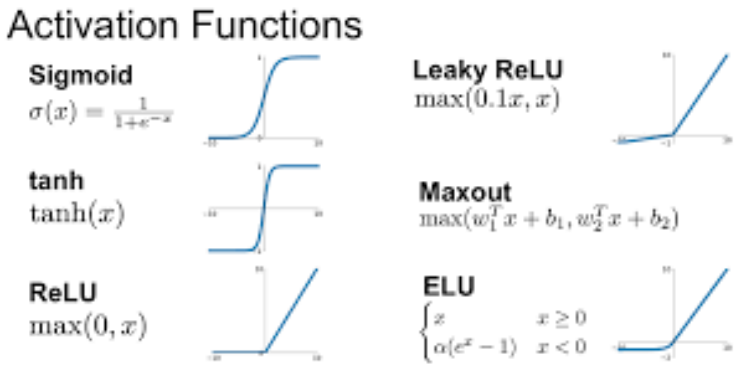
An activation function, most commonly a sigmoid function, just scales the output to be between 0 and 1 again — so it is a logistic function.

$$\textit{sigmoid} = \sigma = \frac{1}{1 + e^{-x}} = \text{number between 0 and 1.}$$



Activation Functions

- We wrap the equation for new neurons with the activation, i.e. multiply summarization of the result of multiplying the weights and activations
  $\sigma(w_1a_1 + w_2a_2 + ... + w_na_n)$ = new neuron.
- Other commonly used activation functions include: tanh, ReLU, leaky ReLU, etc.
- ReLU and leaky ReLU were introduced to overcome the issue of vanishing gradients when the activation function saturates, causing no effective learning.



Bias

Bias is trying to approximate where the value of the new neuron starts to be meaningful. We just need to add or subtract bias from the multiplication of weights and activations.
$\sigma(w_1 a_1 + w_2 a_2 + ... + w_n a_n + b)$ = new neuron.

Cross Entropy Loss

Neural Networks use Cross-Entropy Loss function for Training.

- Each predicted probability is compared to the actual class output value (0 or 1) and a score is calculated that penalizes the probability based on the distance from the expected value.
- The penalty is logarithmic, offering a small score for small differences (0.1 or 0.2) and enormous score for a large difference (0.9 or 1.0).
- Cross-entropy loss is minimized, where smaller values represent a better model than larger values. A model that predicts perfect probabilities has a cross entropy or log loss of 0.0.
- Cross-entropy for a binary or two class prediction problem is actually calculated as the average cross entropy across all examples.

Cross Entropy Loss

- Cross-entropy can be calculated for multiple-class classification.
- The classes have been one hot encoded, meaning that there is a binary feature for each class value and the predictions must have predicted probabilities for each of the classes.
- The cross-entropy is then summed across each binary feature and averaged across all examples in the dataset.

Ensemble Classifier

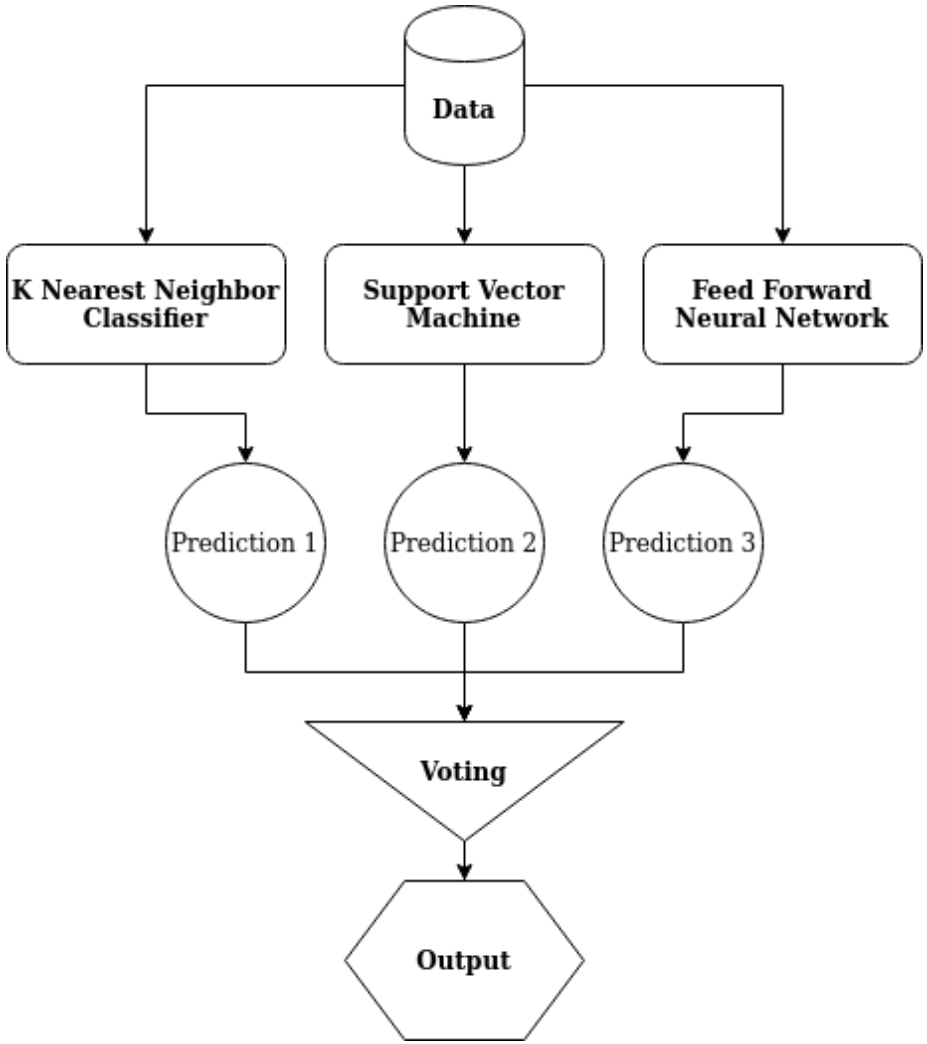


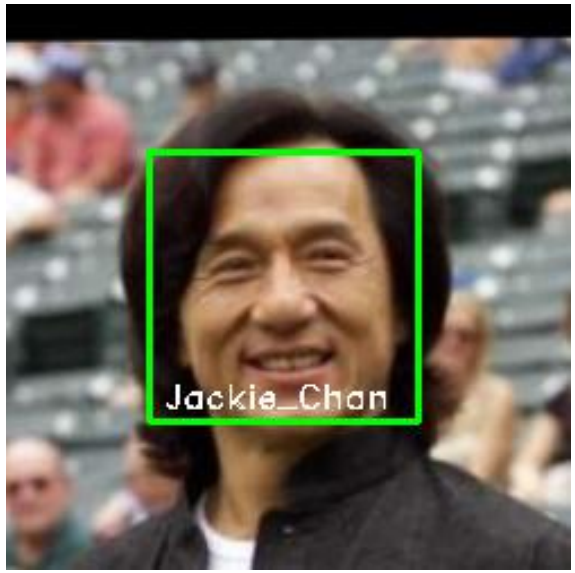
Figure: Ensemble Classifier

Implementation

The system design described was implemented using Python programming language (in Jupyter notebook on Google Colaboratory platform - with GPU hardware accelerator).

Face Detection

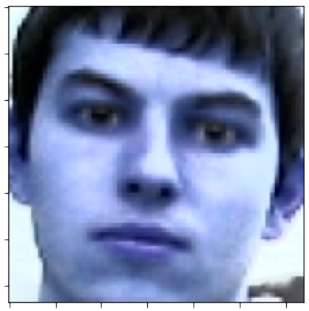
- Achieved using **Viola Jones Algorithm** - OpenCV's Haar Cascade Classifer ("**haarcascade_frontal_alt.xml**")
- OpenCV's "**rectangle**" function used to frame the detected faces
- OpenCV's "**putText**" function used to label the faces as identified by the system
- Crop out the framed (detected) faces for classification



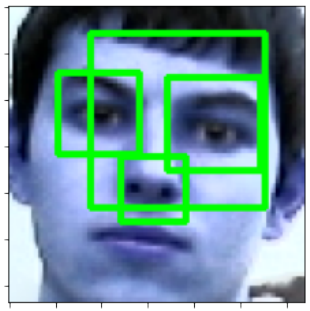
Implementation

Image processing

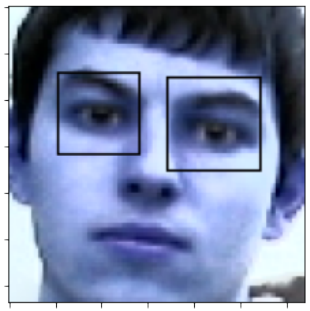
- Resize to common image dimensions
- Align Faces
- Histogram Equalize
- Convert RGB images to Gray scale images
- Linearize



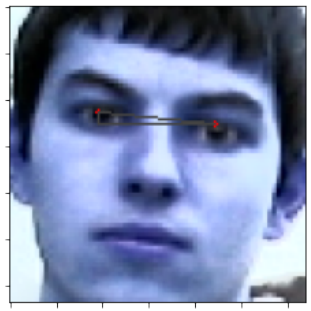
Unaligned



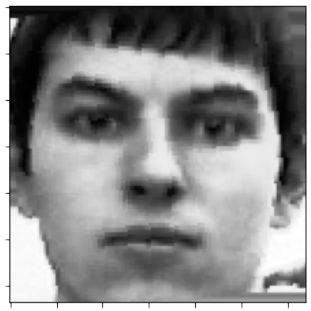
Eye Detection



Best eyes



Rotation Angle



Aligned Face

Implementation

Principal Component Analysis and Data Pre-processing

- Training images linearized and used to determine set of eigenfaces using PCA, with the number of components being a free parameter.
- If no test set available, perform train-test (or validation) split.
- All samples mapped to reduced dimensional eigen-vector space.
- Standardize the both set of samples using mean and variance determined using training samples.

$$\underset{Image-Space}{\begin{pmatrix} z_{11} & z_{12} & \cdots & z_{1n} \\ z_{21} & z_{22} & \cdots & z_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{nn} \end{pmatrix}} \rightarrow \underset{Face-Space}{\begin{pmatrix} z_{11} & \cdots & z_{1n} & \cdots & z_{nn} \end{pmatrix}} \rightarrow \underset{Eigenface-Space}{\begin{pmatrix} Z_1 & Z_2 & \cdots & Z_m \end{pmatrix}}$$

Implementation

Training Classifiers

- **k Nearest Neighbours (kNN)**: No training - direct classification of test (or validation) samples
- **Support Vector Machine (SVM)**: Used grid search to determine best kernel and corresponding parameters using validation set.
- **Feed Forward Neural Network (FFNN)**: Designed and trained a fully connected neural network to classify the test samples. Features:
  1. L2 regularization
  2. batch normalization
  3. dropout
  4. "tanh" activation function
  5. PyTorch - training on GPU (if available)
  6. 5 layers Neural Network
  7. Loss function - cross entropy loss
- **Ensemble Classifier**: Outputs of above classifiers were weighted in the ratio 0.3:0.3:0.4, respectively and obtained final output.

Washington Dataset

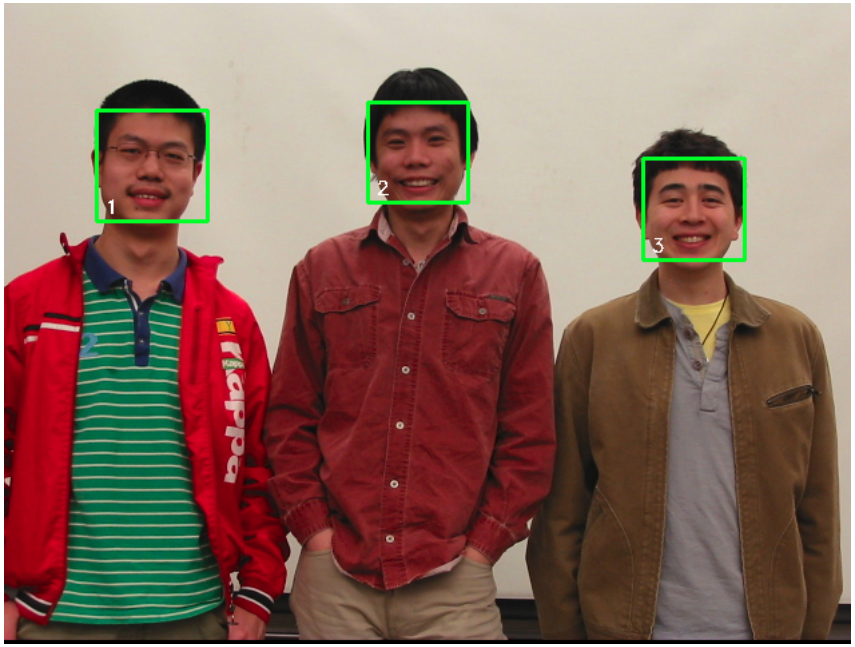


Figure: Output Example 1

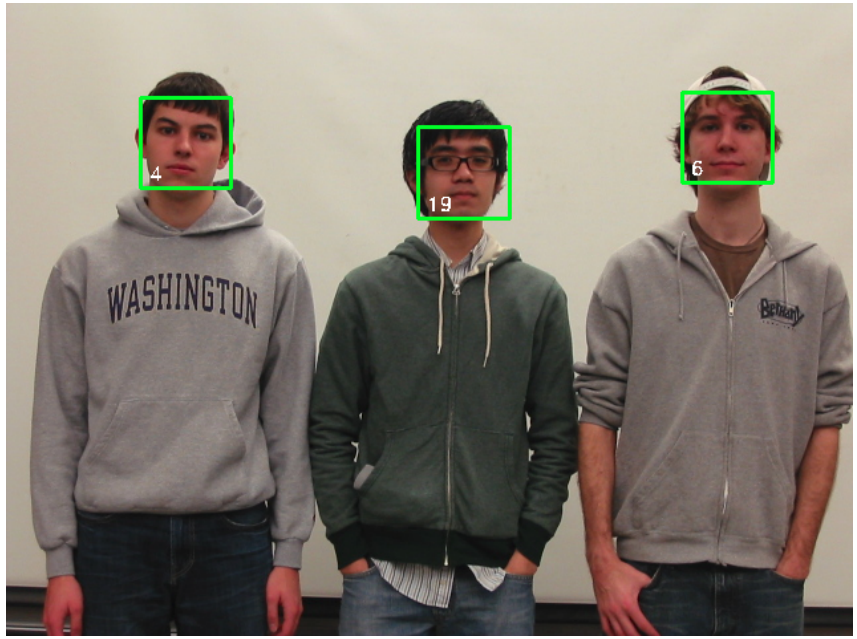


Figure: Output Example 2

Washington Dataset

Bird's Eye View (Flow Diagarm)

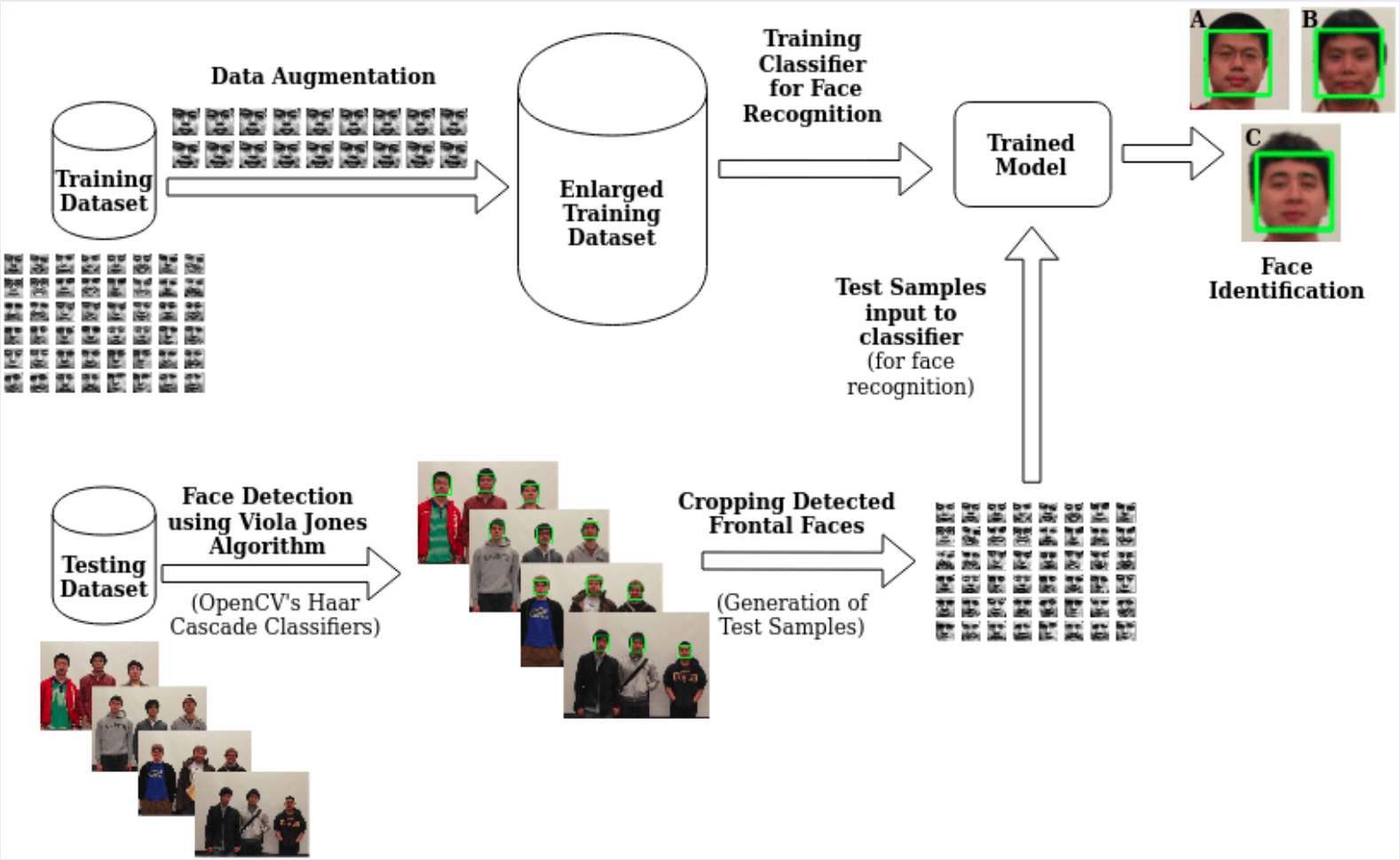
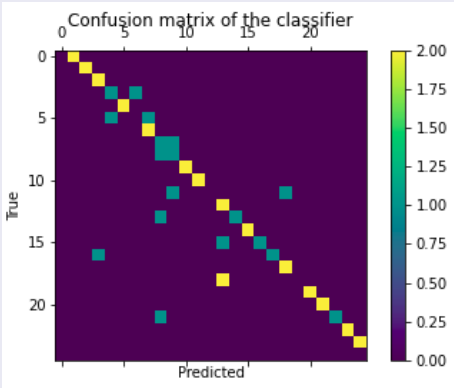


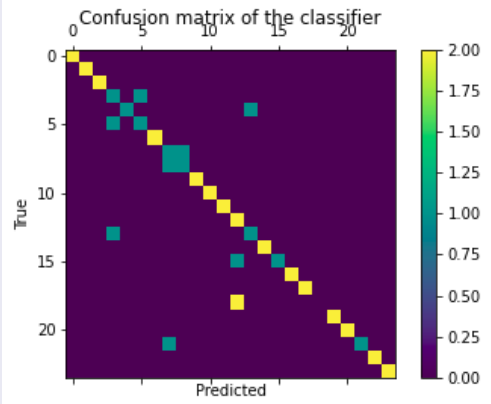
Figure: Flow Diagram

Washington Dataset

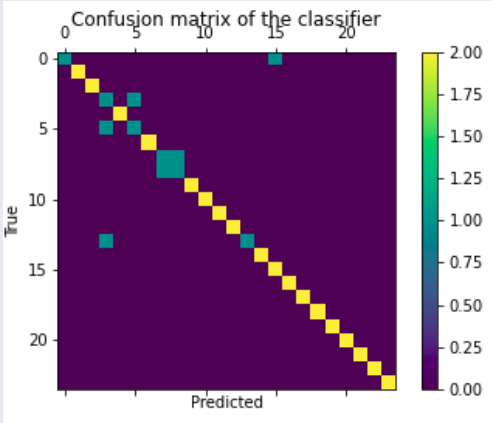
Performance on Test Set



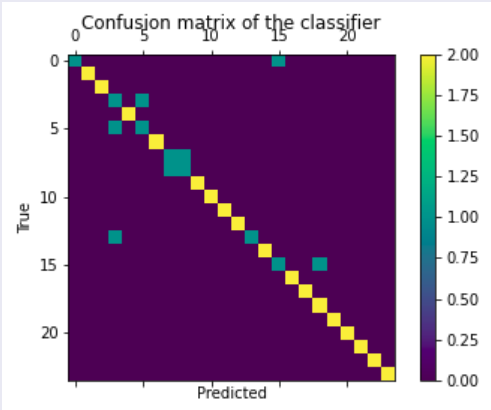
kNN: 72.916%



SVM: 79.167%



FFNN: 91.666%



Ensemble Classifier: 85.416%

Washington Dataset

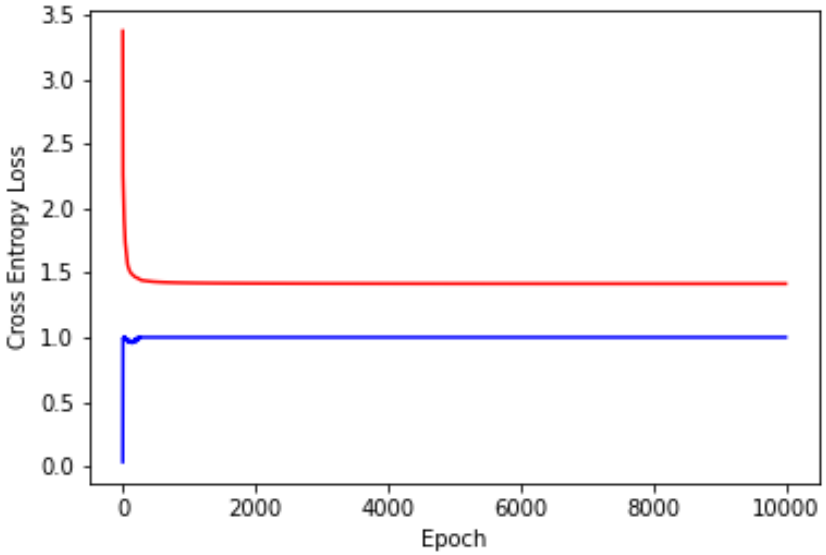


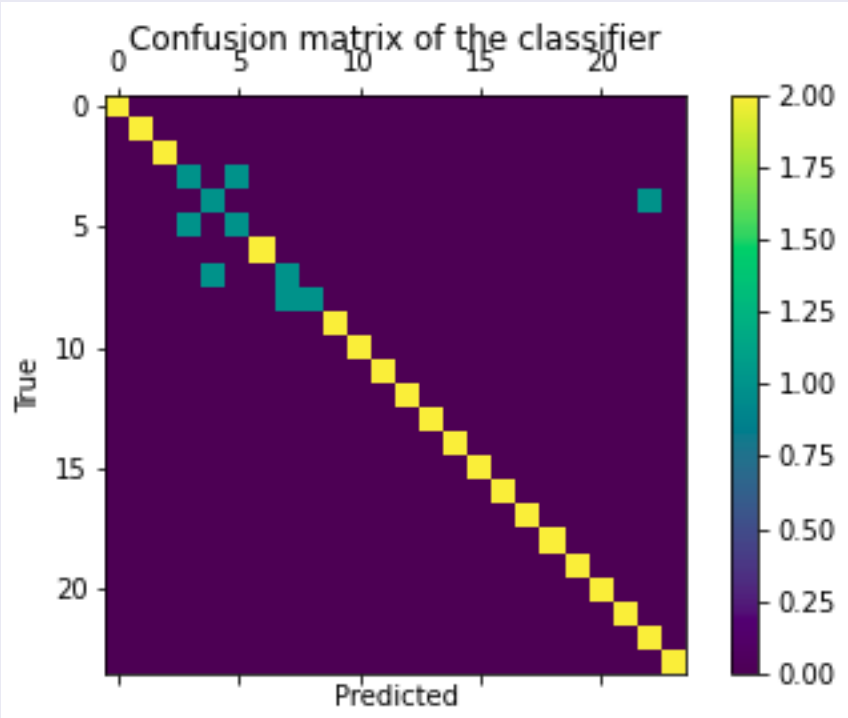
Figure: Loss Curve (Red)
Accuracy (Blue)

```
FFNNetwork_Regularized(
  (net): Sequential(
    (0): RegularizedLinear(in_features=126, out_features=250, bias=True)
    (1): BatchNorm1d(250, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Hakuna_Matata()
    (3): RegularizedLinear(in_features=250, out_features=150, bias=True)
    (4): BatchNorm1d(150, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): Dropout(p=0.2, inplace=False)
    (6): Hakuna_Matata()
    (7): RegularizedLinear(in_features=150, out_features=48, bias=True)
    (8): BatchNorm1d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): Hakuna_Matata()
    (10): RegularizedLinear(in_features=48, out_features=24, bias=True)
    (11): BatchNorm1d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): Hakuna_Matata()
    (13): Softmax(dim=None)
  )
)
```

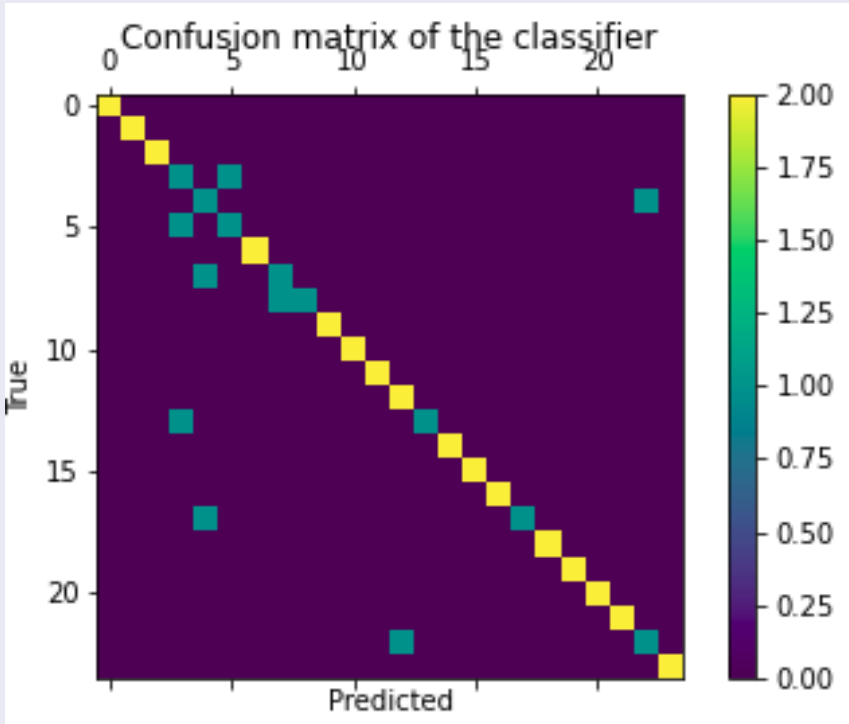
Figure: FFNN architecture

Washington Dataset

Performance on Test set



Generalized FFNN: 87.5%



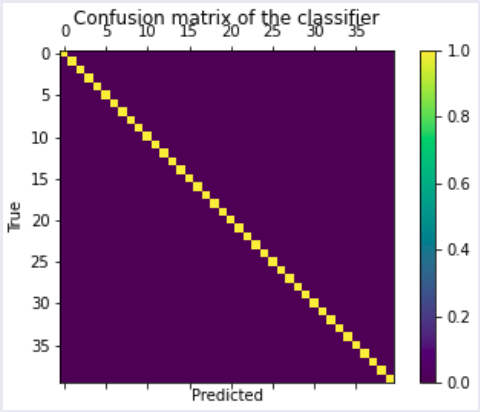
Ensemble classifier (with generalized FFNN): 83.3%

Fetch Olivetti Faces Dataset

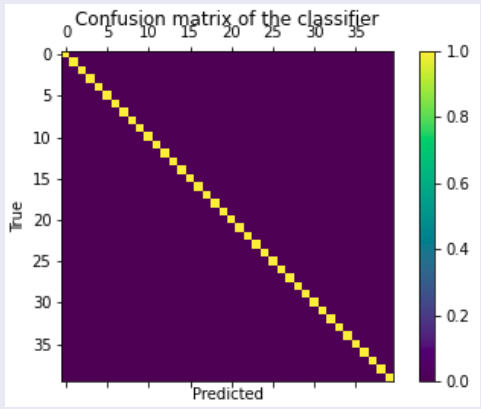
Figure: Samples of Complete Dataset

Fetch Olivetti Faces Dataset

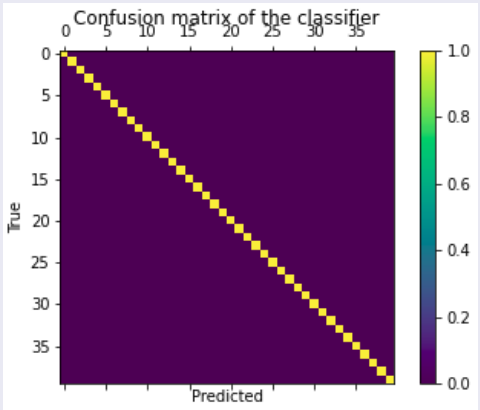
Performance on Validation Set



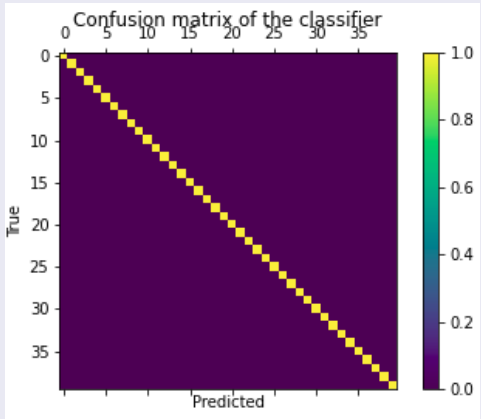
100%



100%



100%



100%

Fetch Olivetti Faces Dataset

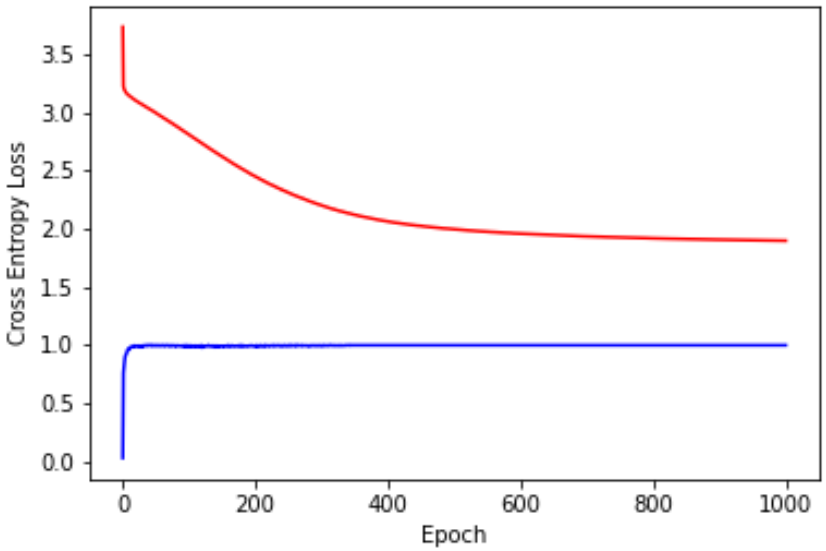


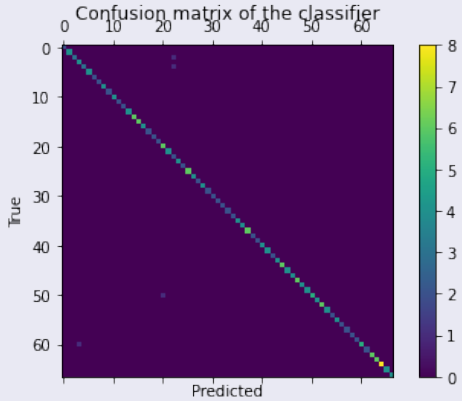
Figure: Loss Curve (Red)
Accuracy (Blue)

```
FFNNetwork_Regularized(
  (net): Sequential(
    (0): RegularizedLinear(in_features=262, out_features=2048, bias=True)
    (1): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Dropout(p=0.1, inplace=False)
    (3): Hakuna_Matata()
    (4): RegularizedLinear(in_features=2048, out_features=1024, bias=True)
    (5): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Hakuna_Matata()
    (7): RegularizedLinear(in_features=1024, out_features=512, bias=True)
    (8): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): Hakuna_Matata()
    (10): RegularizedLinear(in_features=512, out_features=250, bias=True)
    (11): BatchNorm1d(250, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): Hakuna_Matata()
    (13): RegularizedLinear(in_features=250, out_features=93, bias=True)
    (14): BatchNorm1d(93, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): Hakuna_Matata()
    (16): Softmax(dim=None)
  )
)
```

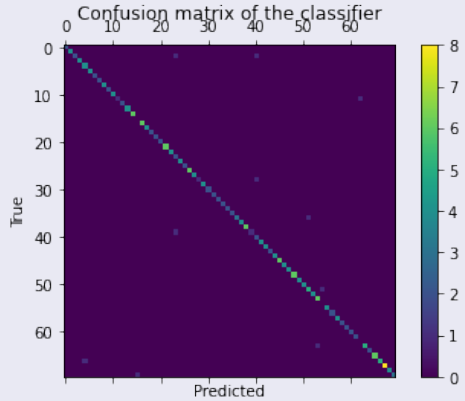
Figure: FFNN architecture

LFW Dataset (Subset)

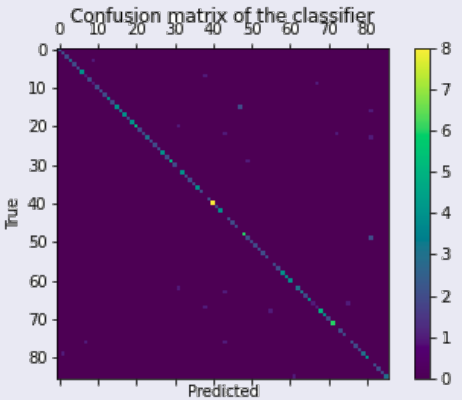
Performance on Validation Set



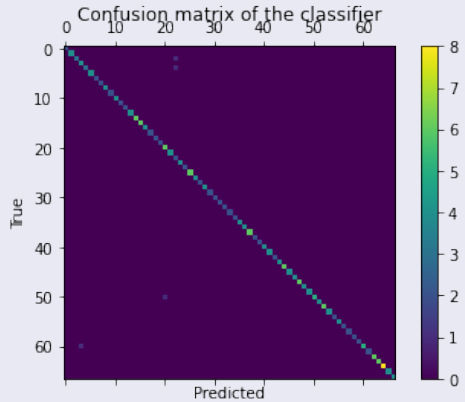
99.14%



94.39%



92.24%



94.83%

LFW Dataset (Subset)

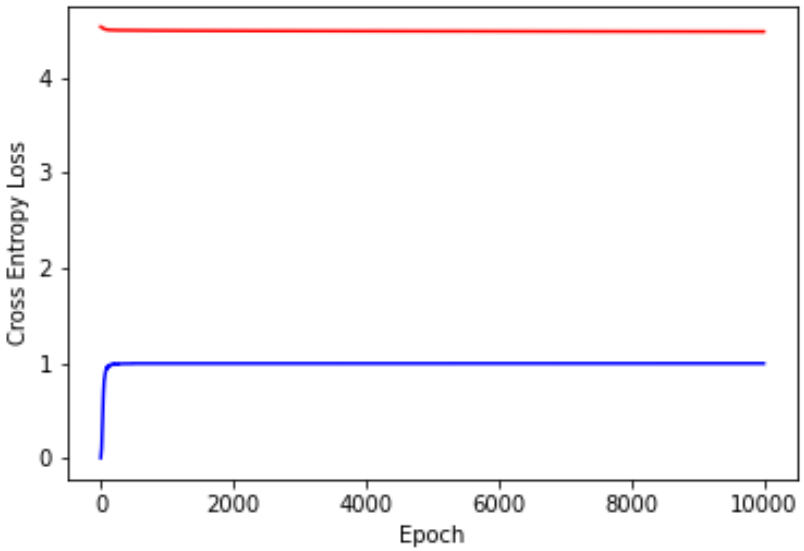


Figure: Loss Curve (Red)
Accuracy (Blue)

```
FFNNetwork_Regularized(
  (net): Sequential(
    (0): RegularizedLinear(in_features=262, out_features=2048, bias=True)
    (1): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Dropout(p=0.1, inplace=False)
    (3): Hakuna_Matata()
    (4): RegularizedLinear(in_features=2048, out_features=1024, bias=True)
    (5): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Hakuna_Matata()
    (7): RegularizedLinear(in_features=1024, out_features=512, bias=True)
    (8): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): Hakuna_Matata()
    (10): RegularizedLinear(in_features=512, out_features=250, bias=True)
    (11): BatchNorm1d(250, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): Hakuna_Matata()
    (13): RegularizedLinear(in_features=250, out_features=93, bias=True)
    (14): BatchNorm1d(93, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): Hakuna_Matata()
    (16): Softmax(dim=None)
  )
)
```

Figure: FFNN architecture

Attendance System

Description

- The trained models of FFNN were saved in a ".txt" file.
- For SVM, the best kernel and corresponding set of parameters were noted and used for classification.
- For kNN, we require the training samples (vectors). For this, the training samples were stored in a ".txt" file using numpy's **savetxt** function and retrieved using numpy's **loadtxt** function.

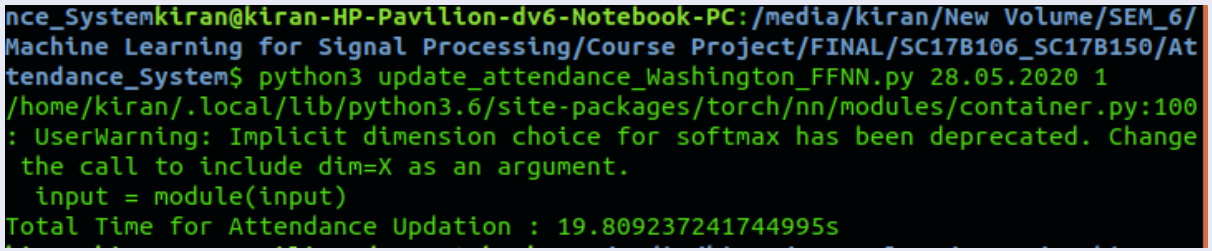


Figure: Attendance system

Attendance System

Inputs to attendance system

- The test images need to be stored in a separate directory.
- The paths to directories consisting of test images, trained model files, and attendance sheet (excel file) need to be input inside the code. It is adviced to have a separate folder created to store the output files as well.
- The only other inputs required by the system now are the **date** (by default, the date of execution of the code) and **key value**.
- Key value should be set to "1" to save output files in unique folder with name corresponding to the date.
- By default, the attendance system stores the latest output files in a temporary folder.

Attendance System

Inputs

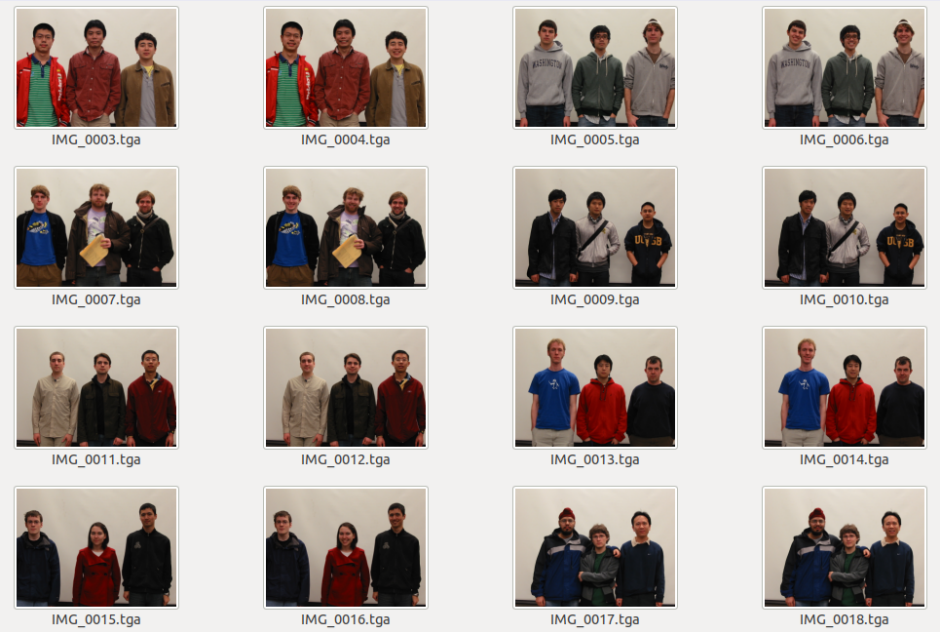


Figure: Input test images from Washington Dataset

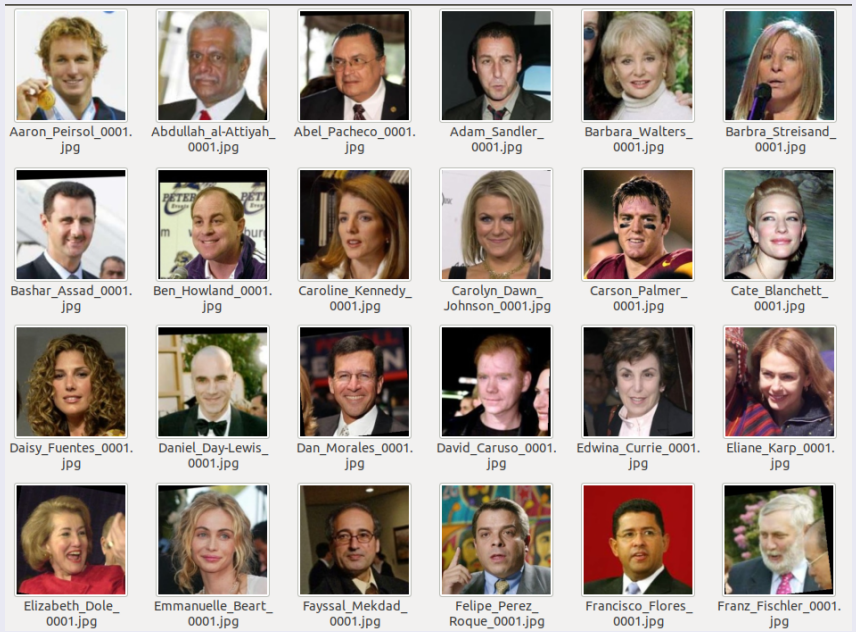


Figure: Input validation images from LFW Dataset (subset)

Attendance System

Two Different Type of Inputs

- The attendance system so designed has two type of inputs that can be given to it.
  1. Only one subject per test image
  2. More than one subjects in a given test image
- For each type of input, there exists two types of codes
  1. one which considers only trained FFNN for identification (suffixed with "_FFNN")
  2. other which considers all three classifier predictions and outputs the prediction of the ensemble classifier (no suffix)

Attendance System

Two Different Systems

- The code "update_attendance_LFW.py" is designed for use when the test set consists of only one subject per test image.

```
kiran@kiran-HP-Pavilion-dv6-Notebook-PC:/media/kiran/New Volume/SEM_6/Machine Learnin
g for Signal Processing/Course Project/FINAL/SC17B106_SC17B150/Attendance_System$ pyt
hon3 update_attendance_LFW_FFNN.py 27/05/2020
/home/kiran/.local/lib/python3.6/site-packages/torch/nn/modules/container.py:100: Use
rWarning: Implicit dimension choice for softmax has been deprecated. Change the call
to include dim=X as an argument.
  input = module(input)
Total Time for Attendance Updation : 25.90116262435913s
```

Figure: update_attendance_LFW_FFNN.py

```
kiran@kiran-HP-Pavilion-dv6-Notebook-PC:/media/kiran/New Volume/SEM_6/Mach
ine Learning for Signal Processing/Course Project/FINAL/SC17B106_SC17B150/
Attendance_System$ python3 update_attendance_LFW.py
/home/kiran/.local/lib/python3.6/site-packages/torch/nn/modules/container.
py:100: UserWarning: Implicit dimension choice for softmax has been deprec
ated. Change the call to include dim=X as an argument.
  input = module(input)
Total Time for Attendance Updation : 104.05464720726013s
```

Figure: update_attendance_LFW.py

Attendance System

Two Different Systems

- The code "update_attendance_Washington.py" is designed for use on test set with more than one test subjects per image.

```
kiran@kiran-HP-Pavilion-dv6-Notebook-PC:/media/kiran/New Volume/SEM_6/Machine Learnin
g for Signal Processing/Course Project/FINAL/SC17B106_SC17B150/Attendance_System$ pyt
hon3 update_attendance_Washington_FFNN.py
/home/kiran/.local/lib/python3.6/site-packages/torch/nn/modules/container.py:100: Use
rWarning: Implicit dimension choice for softmax has been deprecated. Change the call
to include dim=X as an argument.
  input = module(input)
Total Time for Attendance Updation : 16.964905500411987s
```

Figure: update_attendance_Washington_FFNN.py

```
kiran@kiran-HP-Pavilion-dv6-Notebook-PC:/media/kiran/New Volume/SEM_6/Machine Learnin
g for Signal Processing/Course Project/FINAL/SC17B106_SC17B150/Attendance_System$ pyt
hon3 update_attendance_Washington.py 27/05/2020
/home/kiran/.local/lib/python3.6/site-packages/torch/nn/modules/container.py:100: Use
rWarning: Implicit dimension choice for softmax has been deprecated. Change the call
to include dim=X as an argument.
  input = module(input)
Total Time for Attendance Updation : 27.53991961479187s
```

Figure: update_attendance_Washington.py

Attendance System

Outputs

Output of the attendance system consists of the name tagged test images (for physical verification) and updated attendance sheet.

Figure: Output files of Washington Test Dataset

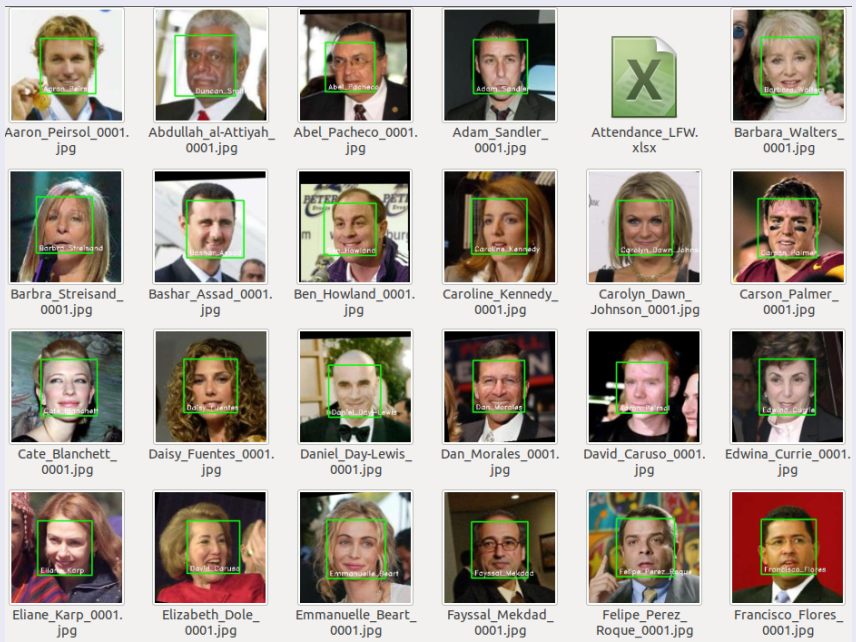
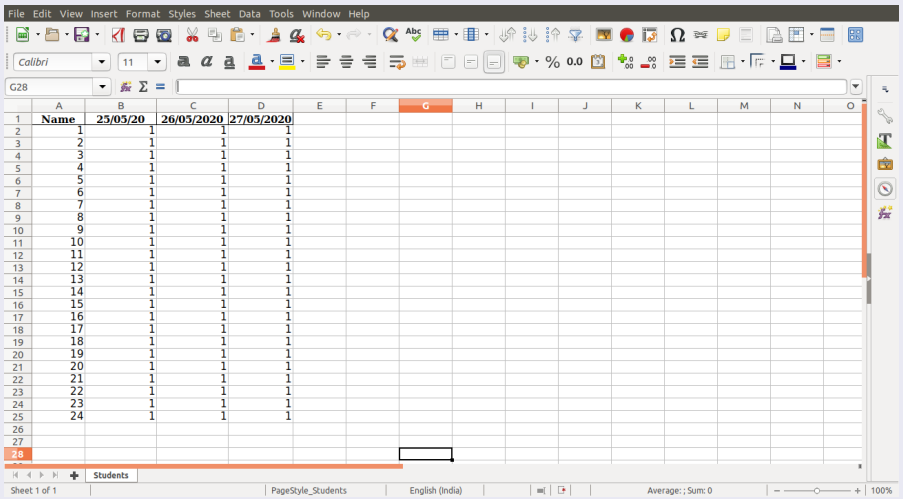


Figure: Output files of LFW Dataset (subset) validation set

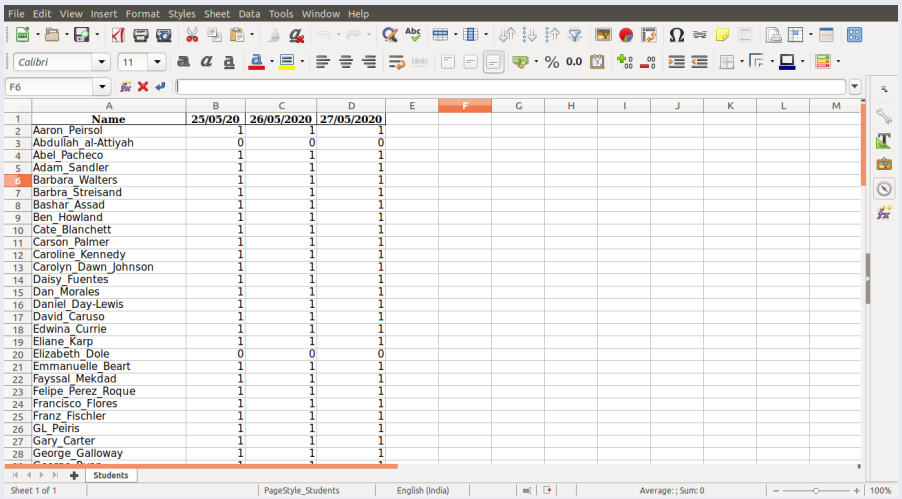
Attendance System

Attendance File Output



Name	25/05/20	26/05/2020	27/05/2020
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1
7	1	1	1
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	1
13	1	1	1
14	1	1	1
15	1	1	1
16	1	1	1
17	1	1	1
18	1	1	1
19	1	1	1
20	1	1	1
21	1	1	1
22	1	1	1
23	1	1	1
24	1	1	1

Figure: Attendance updation of subjects in Washington Test Dataset



Name	25/05/20	26/05/2020	27/05/2020
Aaron_Peirsol	1	1	1
Abdullah_al-Attiyah	0	0	0
Abel_Pacheco	1	1	1
Adam_Sandler	1	1	1
Barbara_Walters	1	1	1
Barbra_Streisand	1	1	1
Bashar_Assad	1	1	1
Ben_Howland	1	1	1
Cate_Blanchett	1	1	1
Carson_Palmer	1	1	1
Caroline_Kennedy	1	1	1
Carolyn_Dawn_Johnson	1	1	1
Daisy_Fuentes	1	1	1
Dan_Morales	1	1	1
Daniel_Day-Lewis	1	1	1
David_Caruso	1	1	1
Edwina_Currie	1	1	1
Eliane_Karp	1	1	1
Elizabeth_Dole	0	0	0
Emmanuelle_Beart	1	1	1
Fayssal_Mekdad	1	1	1
Felipe_Perez_Roque	1	1	1
Francisco_Flores	1	1	1
Franz_Fischler	1	1	1
GL_Peiris	1	1	1
Gary_Carter	1	1	1
George_Galloway	1	1	1

Figure: Attendance updation of subjects in LFW Dataset (subset) validation set

Attendance System

Setup

- Collect individual photographs of each subject whose attendance is to be marked
- Ensure that the collected images (training samples) contain various facial expressions of each person.
- This can be done by placing the subject in front of the web cam with a plain background and capturing 10 to 20 images per person.
- Then, use the given source codes to load the training samples and train the classifiers to generate final usable model.

Attendance System

Limitations and plausible solutions

- **Non-Detection of Faces:** In case the attendance system does not detect all the faces, then the scaleFactor and minNeighbors arguments of haar cascade classifier's "detectMultiScale" function need to be reduced appropriately till all the faces are detected.

Attendance System

Limitations and plausible solutions

- **Inaccurate Identification:**
  1. Increase the number of training samples and re-train the model using the source code(s).
  2. Ensure good clarity of test images - good illumination, alignment, and facial expression similar to those in training samples.
  3. Change the weightage for the classifiers in the ensemble classifier (if using it).
  4. While the best model is still to be obtained, physically verify the system output and use the same for re-training the model.

Attendance System

Limitations and plausible solutions

- **Problem of False Positives:** There can be cases where the classifier mis-classifies a person and it turns out to be a situation where an absentee is marked present. To overcome this issue to certain extent,
  1. Increase the number of testing samples per person, i.e., induce redundancy in the test set (say 3 test images per person).
  2. Perform a voting and mark a person as present if identified in at least 2 out of 3 test images, else, mark as absent.

The above measures are suggested under the assumption that the model being used is well trained (with validation accuracy close to greater than 90%)

Conclusion

Conclusion

- The presented system successfully detects faces from static images and identifies the people with appropriate training set.
- Higher the variability of the face images (of each subject) used for training, better is the performance of the classifiers designed.
- SVM and FFNN classifiers are robust while the kNN aces only when degree of similarity of train and test samples is high.
- The neural network architecture depends on the number of subjects to be classified. Hence, no generalized neural network could be designed.
- The biometric attendance system has been successfully designed which can perform on static images with both single subject and multiple subjects per image.

Future Enhancement

Future Enhancement

- The existing work can be improvised for commercial usage in biometric face recognition based systems (such as the attendance system presented here).
- Further, the same system can be designed using state of the art technologies and methods, such as Convolutional Neural Networks which takes care of the task of feature engineering.
- This can be followed by a set of fully connected layer for classification task. Some of the CNN architectures which are currently used for similar applications are: LeNet, AlexNet, GoogLe Net, Resnet, VGG-16, VGG-19, etc., by altering the input and output layers.
- Also better algorithms and methods needs to be designed to overcome the limitations of the build attendance system (mainly, the issue of false positives).

References

- *https://github.com/parulnith/Face-Detection-in-Python-using-OpenCV/blob/master/Face*
- *https://www.datacamp.com/community/tutorials/face-detection-python-opencv*
- *https://arxiv.org/pdf/1901.02452.pdf*
- *http://cmusatyalab.github.io/openface/*
- *http://www.scholarpedia.org/article/K-nearest_neighbor*
- *https://towardsdatascience.com/a-friendly-introduction-to-support-vector-machines-svm-925b68c5a079*
- *https://mlfromscratch.com/neural-networks-explained/*
- *https://github.com/contail/Face-Alignment-with-OpenCV-and-Python/blob/master/align_faces.py*
- *https://sefiks.com/2020/02/23/face-alignment-for-face-recognition-in-python-within-opencv/*


thank
you!