Mohammed Kamran Syed
50289322
msyed3@buffalo.edu
May 13, 2019

Face Detection using Viola Jones algorithm

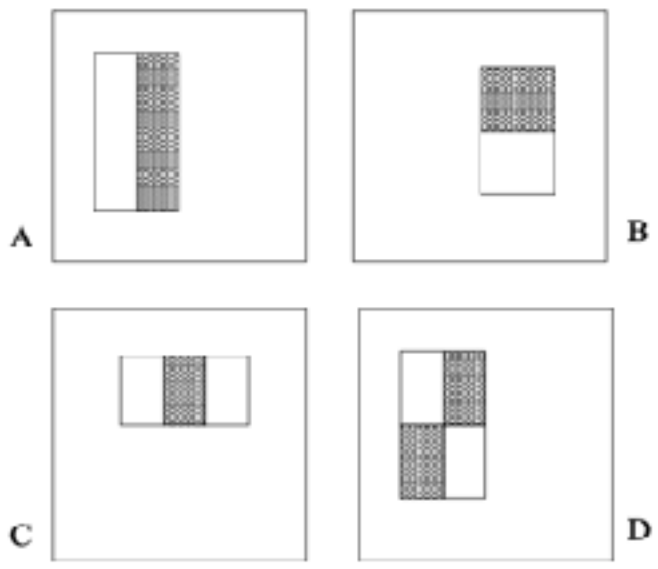
The Overview:

Viola Jones algorithm is based on a machine learning approach to real time object detection. Though the training is slow, it achieves very high detection rates. It uses different weak classifiers (which it creates during the training phase), each looking at a different portion of the image. Each weak classifier is less accurate but on combining all we get a strong classifier.

The algorithm incorporates three important parts:
1. Haar Features
2. Integral Image
3. AdaBoost
4. Attentional Cascade

1. Haar Features:

Features in Viola Jones algorithm are a set of rectangular regions which are marked positive and negative. These features are applied on the image and value is calculated which is the sum of positive regions minus the sum of negative regions. Each of these Haar features represent an area on the face which when applied on the specific area of the face of an image give highest value. The paper describes the following four set of features:



The shaded region above is the positive region and the unshaded one is negative. Feature value is a region is calculated as sum of pixels in shaded region minus sum of pixels in unshaded region.

2. Integral Image:

For calculating the Haar features as shown above, it takes enormous amount of computation (>160K for 24X24 image). Thus viola jones came up with an intermediate representation of an image called as an integral image. The value of integral image at any position x, y is the sum of values of pixels to the left and above the given position x, y.

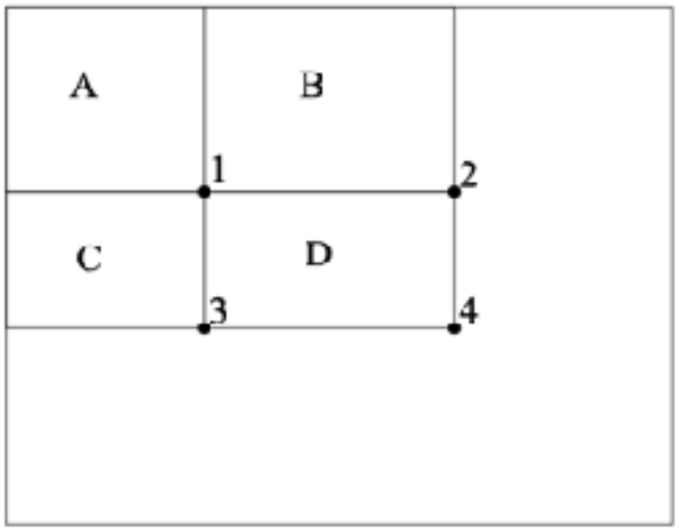
$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

ii(x, y) in the above formula represents value of the integral image at x, y which can be calculated as follows:

$$\begin{aligned} s(x,y) &= s(x,y-1) + i(x,y) \\ ii(x,y) &= ii(x-1,y) + s(x,y) \end{aligned}$$

Where s(x, y) is cumulative row sum with s(x, -1) = 0 and ii(-1, y) = 0.

Using this integral image, rectangular sum of any region can be easily calculated as shown below:

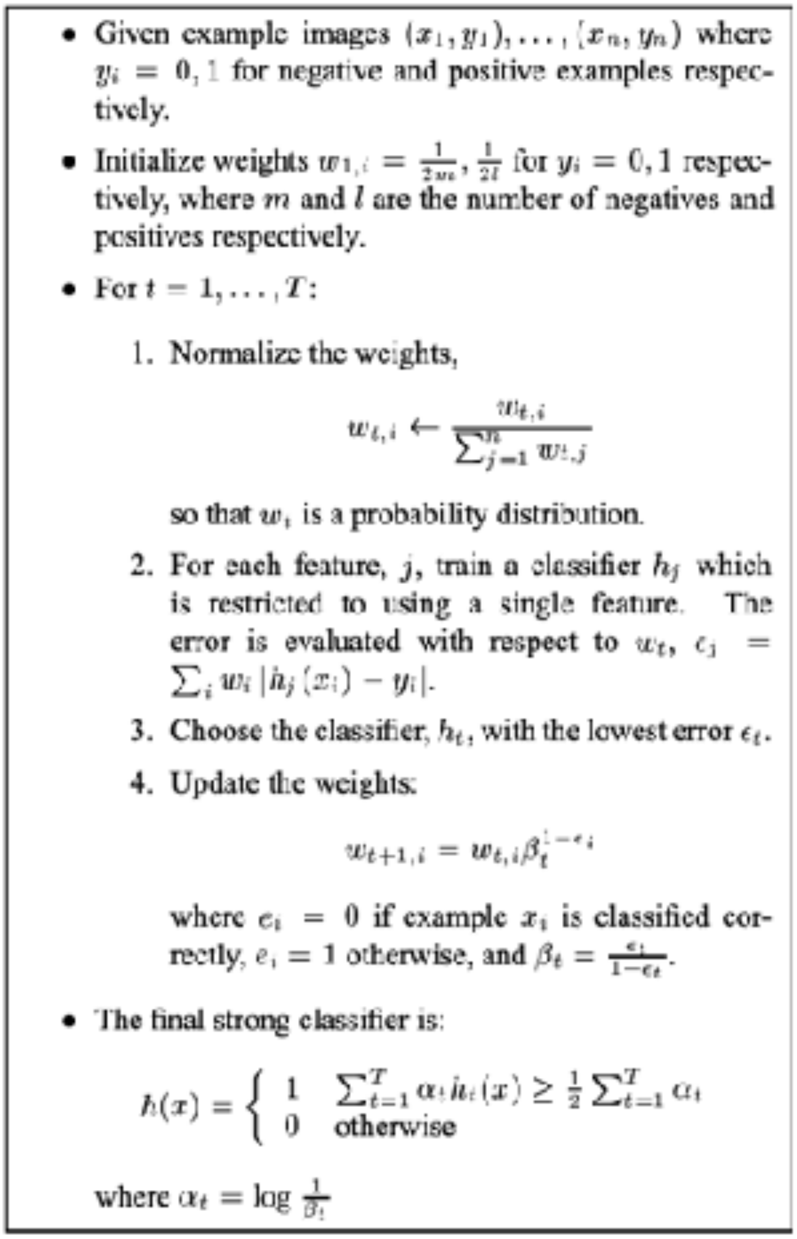


Value at point 1 = A
Value at point 2 = A + B
Value at point 3 = A + C
Value at point 4 = A + B + C + D

Sum at D = 4 + 1 - (2 + 3)

3. AdaBoost:

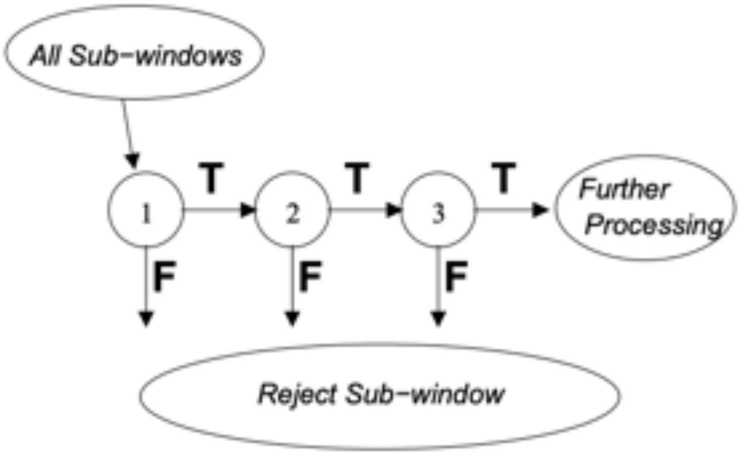
Boosting is a learning scheme that combines weak classifiers into a more accurate ensemble classifier. The idea behind it is for each successor classifier correct the mistakes of its previous classifier. It does this by assigning a weight to each training example, training the classifier, choosing the best classifier and updating the weights according to the error of the classifier. The paper describes the overall training algorithm as shown below:



- Given example images $(x_1, y_1), \dots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \dots, T$:
  1. Normalize the weights,
     $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
     so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \, |h_j(x_i) - y_i|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
     $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
     where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
  $$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
  where $\alpha_t = \log \frac{1}{\beta_t}$

4. The Attentional Cascade:

To improve the performance of face detection, the algorithm makes use of a series of weak classifiers which at very stage reject non faces allowing only possible faces to go through to the next round. A positive response from the first classifier triggers evaluation of a second (more complex) classifier and so on. A negative outcome however leads to immediate rejection.



Implementing the algorithm:

The following steps were incorporated in preparing, training and testing the classifier:

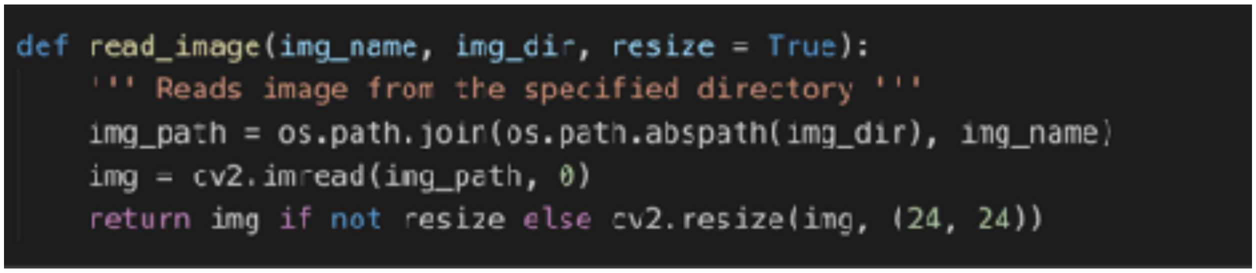
1. Training Data Cleansing:

The training data set used for this project is the FDD dataset found here: http://vis-www.cs.umass.edu/fddb/. The face annotations for the above dataset is also available here: http://vis-www.cs.umass.edu/fddb/FDDB-folds.tgz. A preprocess script (included with the project code) was used to extract out faces using the face annotations file for the training purpose. From the same dataset, a set of non face images (background images) were also extracted. This resulted in about 5000 face images and over 10000 non face images ready for training.

Sample Faces:

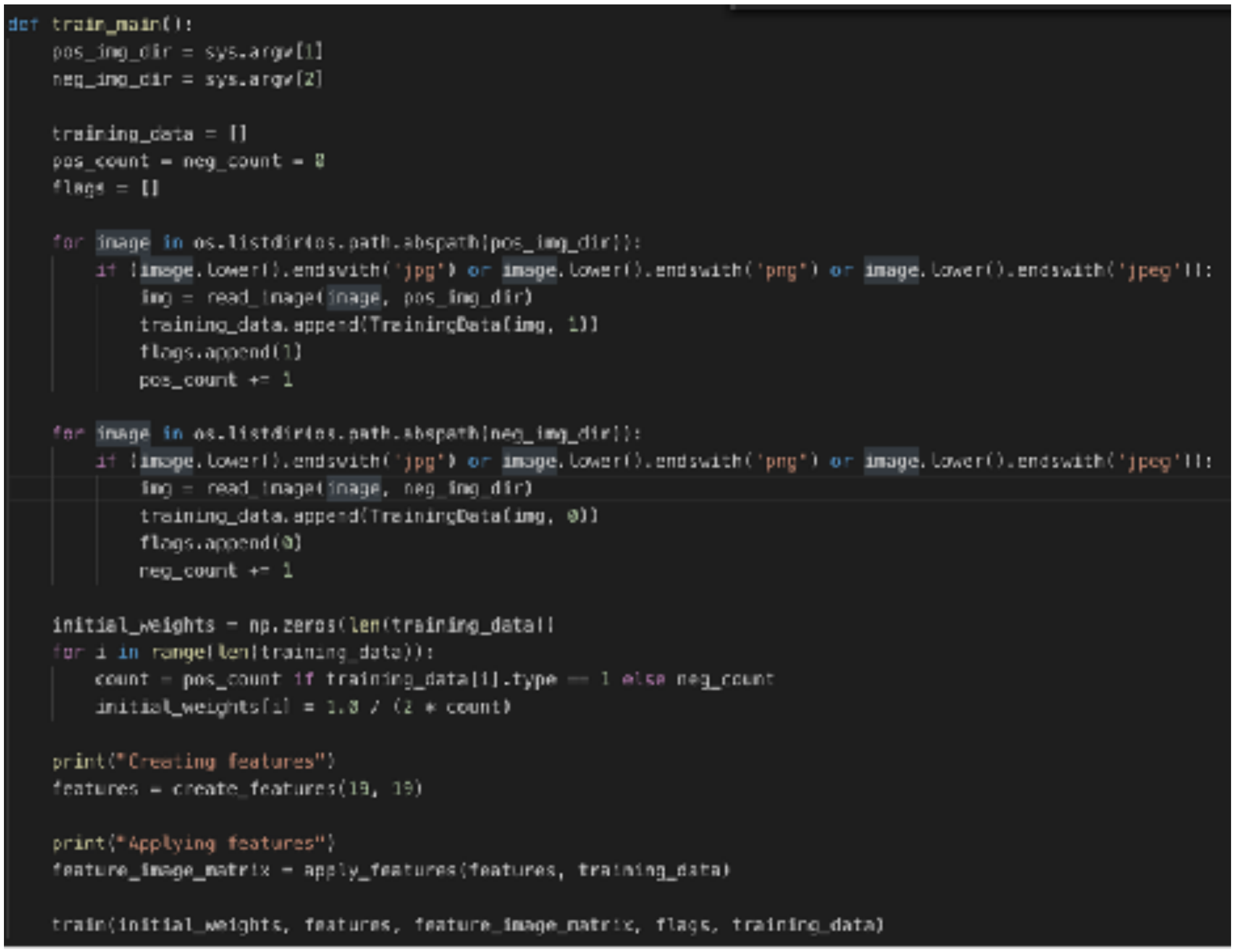
2. Training the classifier:

We start training our classifier by giving the faces and non faces data (generated in the previous step). As per the paper, the classifier has to be trained with 24X24 sized images but due to lack of computation power, our classifier was trained with images of size 19X19. Therefore, every image was read and resized to 19X19 size for training purpose.



```
def read_image(img_name, img_dir, resize = True):
    ''' Reads image from the specified directory '''
    img_path = os.path.join(os.path.abspath(img_dir), img_name)
    img = cv2.imread(img_path, 0)
    return img if not resize else cv2.resize(img, (24, 24))
```

The main training function looks like this:

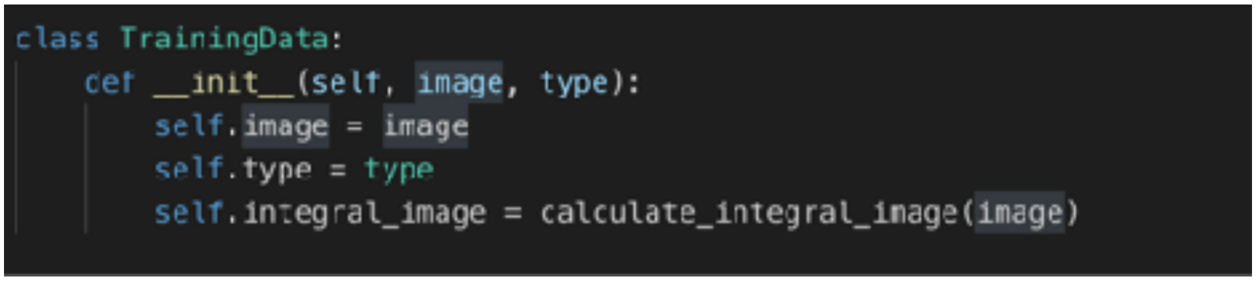


```
def train_main():
    pos_img_dir = sys.argv[1]
    neg_img_dir = sys.argv[2]

    training_data = []
    pos_count = neg_count = 0
    flags = []

    for image in os.listdir(os.path.abspath(pos_img_dir)):
        if (image.lower().endswith('jpg') or image.lower().endswith('png') or image.lower().endswith('jpeg')):
            img = read_image(image, pos_img_dir)
            training_data.append(TrainingData(img, 1))
            flags.append(1)
            pos_count += 1

    for image in os.listdir(os.path.abspath(neg_img_dir)):
        if (image.lower().endswith('jpg') or image.lower().endswith('png') or image.lower().endswith('jpeg')):
            img = read_image(image, neg_img_dir)
            training_data.append(TrainingData(img, 0))
            flags.append(0)
            neg_count += 1

    initial_weights = np.zeros(len(training_data))
    for i in range(len(training_data)):
        count = pos_count if training_data[i].type == 1 else neg_count
        initial_weights[i] = 1.0 / (2 * count)

    print("Creating features")
    features = create_features(19, 19)

    print("Applying features")
    feature_image_matrix = apply_features(features, training_data)

    train(initial_weights, features, feature_image_matrix, flags, training_data)
```

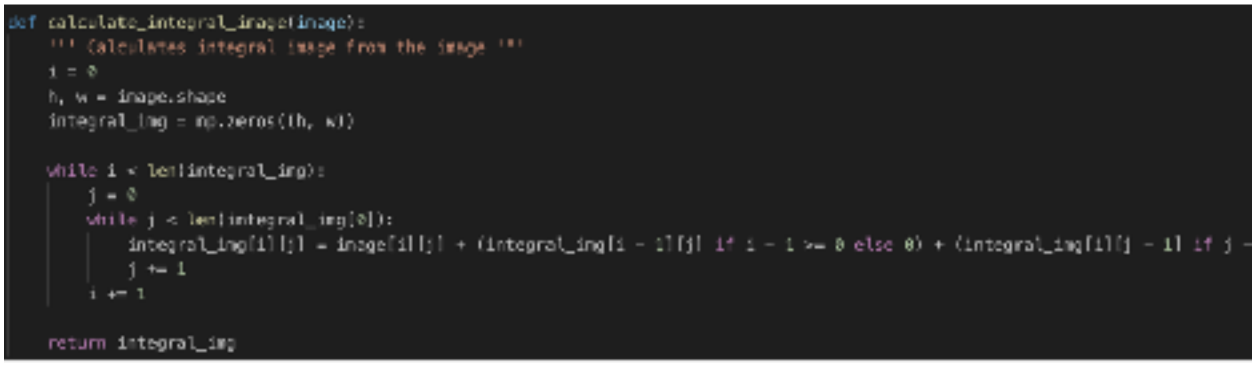
We provide our program with positive and negative image directories as command line arguments. The program creates the training dataset by reading all images (resized to 19X19).

a. Computing Integral Image:

While reading the images, we also compute the integral images of the original images for training purposes and store it in the object of type TrainingData.



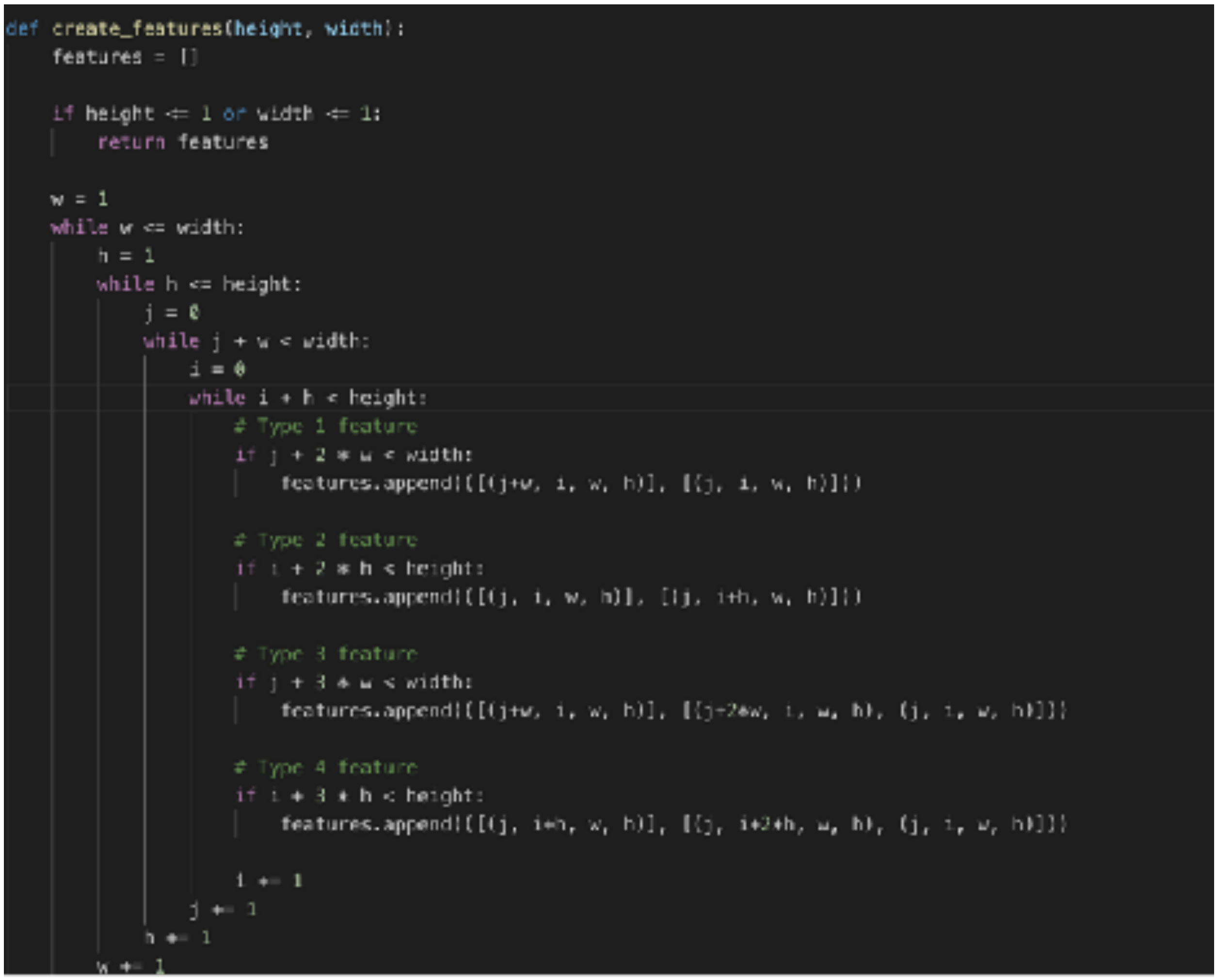
```python
class TrainingData:
    def __init__(self, image, type):
        self.image = image
        self.type = type
        self.integral_image = calculate_integral_image(image)
```



```python
def calculate_integral_image(image):
    ''' Calculates integral image from the image '''
    i = 0
    h, w = image.shape
    integral_img = np.zeros((h, w))

    while i < len(integral_img):
        j = 0
        while j < len(integral_img[0]):
            integral_img[i][j] = image[i][j] + (integral_img[i - 1][j] if i - 1 >= 0 else 0) + (integral_img[i][j - 1] if j -
            j += 1
        i += 1

    return integral_img
```

b. Creating Haar Features:

The next step is to create all possible features (4 types as given in the paper) of all sizes from 1 to 19. The following function pre-computes all features of different sizes:



```python
def create_features(height, width):
    features = []

    if height <= 1 or width <= 1:
        return features

    w = 1
    while w <= width:
        h = 1
        while h <= height:
            j = 0
            while j + w < width:
                i = 0
                while i + h < height:
                    # Type 1 feature
                    if j + 2 * w < width:
                        features.append(([(j+w, i, w, h)], [(j, i, w, h)]))

                    # Type 2 feature
                    if i + 2 * h < height:
                        features.append(([(j, i, w, h)], [(j, i+h, w, h)]))

                    # Type 3 feature
                    if j + 3 * w < width:
                        features.append(([(j+w, i, w, h)], [(j+2*w, i, w, h), (j, i, w, h)]))

                    # Type 4 feature
                    if i + 3 * h < height:
                        features.append(([(j, i+h, w, h)], [(j, i+2*h, w, h), (j, i, w, h)]))

                    i += 1
                j += 1
            h += 1
        w += 1
```

c. Setting Initial weights:

The next step is to set initial weights for the training data. The weight is set as

$$w_i = \begin{cases} \frac{1}{2p} & \text{if } x = 1, \\ \frac{1}{2n} & \text{if } x = 0 \end{cases}$$

Where x is the image type (positive or negative) and p and n are total positive and negative images in the dataset.

d. Applying Features:

The next step is to apply all the features generated in Step b to all the test images.

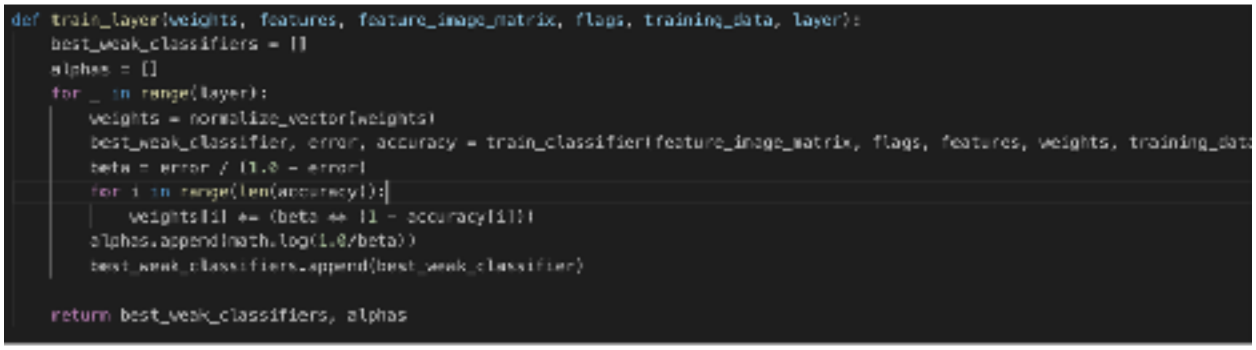
e. Training the Classifier:

The main aim of the algorithm is to generate a set of weak classifiers, each of which is associated with a single feature and has an associated threshold and polarity value which is used to classify a face or a non face.

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

Here p is the polarity of the weak classifier and f(x) is the feature value and theta is the threshold. The classifier outputs 1 if it thinks it as a face or 0 for non face.

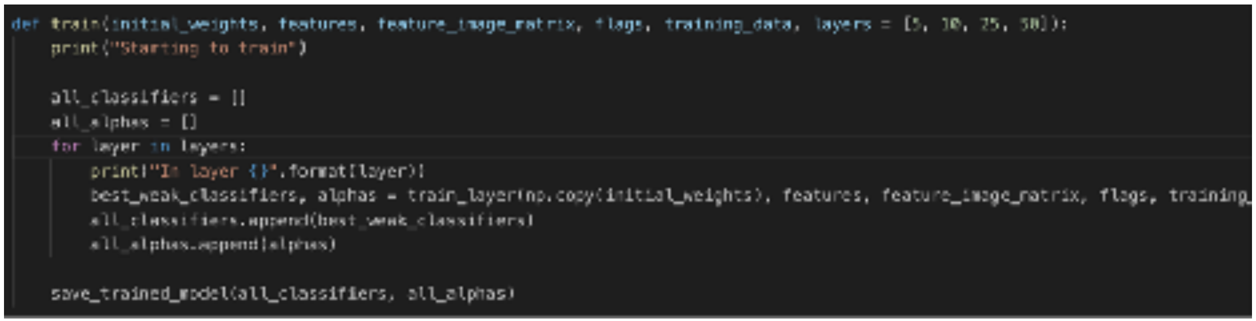
During the training phase, we start off by giving equal weights to all classifiers and then during each round of training, we find the weak classifier with lowest training error and raise the weights of training examples misclassified by current weak classifier.



```
def train_layer(weights, features, feature_image_matrix, flags, training_data, layer):
    best_weak_classifiers = []
    alphas = []
    for _ in range(layer):
        weights = normalize_vector(weights)
        best_weak_classifier, error, accuracy = train_classifier(feature_image_matrix, flags, features, weights, training_dat
        beta = error / (1.0 - error)
        for i in range(len(accuracy)):
            weights[i] *= (beta ** (1 - accuracy[i]))
        alphas.append(math.log(1.0/beta))
        best_weak_classifiers.append(best_weak_classifier)

    return best_weak_classifiers, alphas
```

f. Attentional Cascade:

Improve the speed of the classifier, we can add layers of Weak classifiers as discussed in the overview of the algorithm.

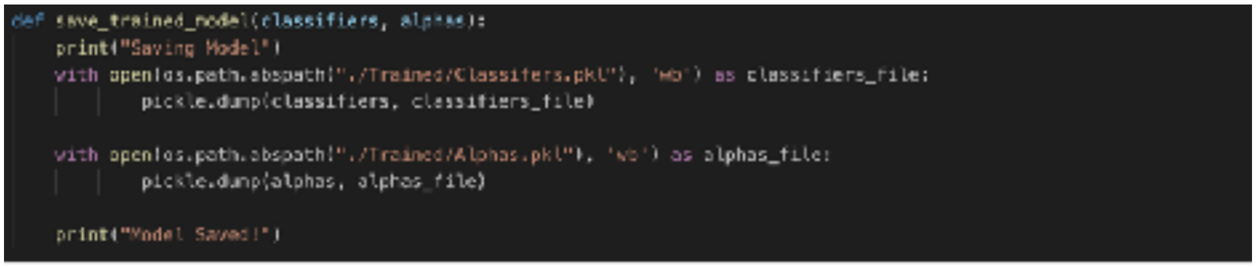


```
def train(initial_weights, features, feature_image_matrix, flags, training_data, layers = [5, 10, 25, 50]):
    print("Starting to train")

    all_classifiers = []
    all_alphas = []
    for layer in layers:
        print("In layer {}".format(layer))
        best_weak_classifiers, alphas = train_layer(np.copy(initial_weights), features, feature_image_matrix, flags, training_
        all_classifiers.append(best_weak_classifiers)
        all_alphas.append(alphas)

    save_trained_model(all_classifiers, all_alphas)
```

As shown above, we add layers of 5, 10, 25 and 50 weak classifiers to our training model to improve the performance and accuracy.

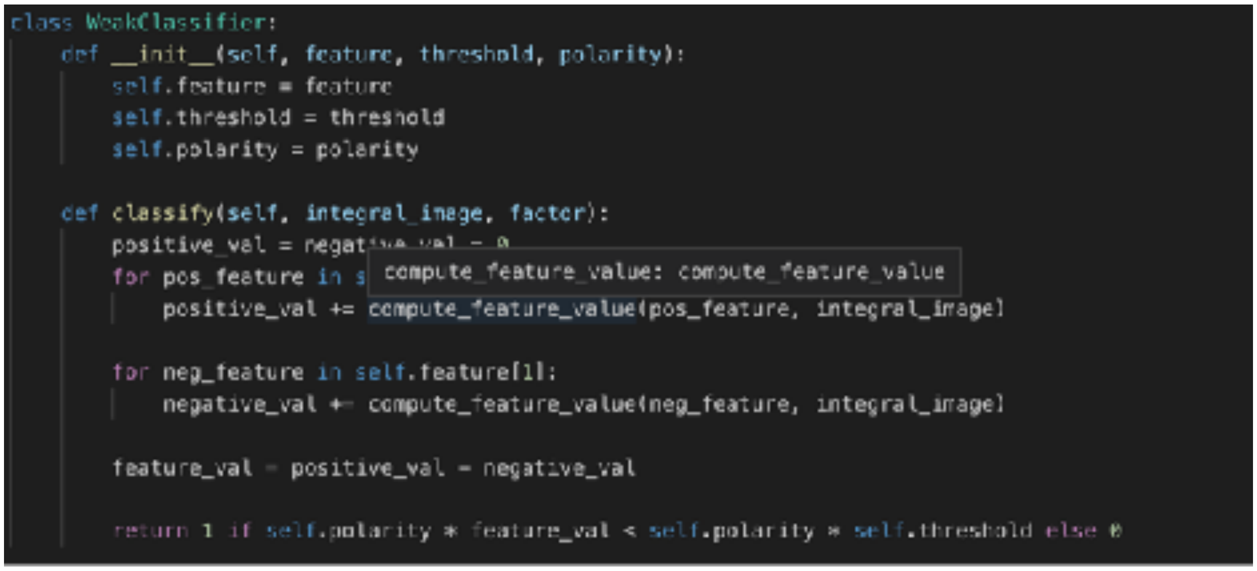
g. Saving the trained model:

The final strong classifier is a set of weak classifiers along with their alpha weights. These are saved in serialized form using Python's pickle library.



```python
def save_trained_model(classifiers, alphas):
    print("Saving Model")
    with open(os.path.abspath("./Trained/Classifers.pkl"), 'wb') as classifiers_file:
            pickle.dump(classifiers, classifiers_file)

    with open(os.path.abspath("./Trained/Alphas.pkl"), 'wb') as alphas_file:
            pickle.dump(alphas, alphas_file)

    print("Model Saved!")
```

3. Testing the classifier:

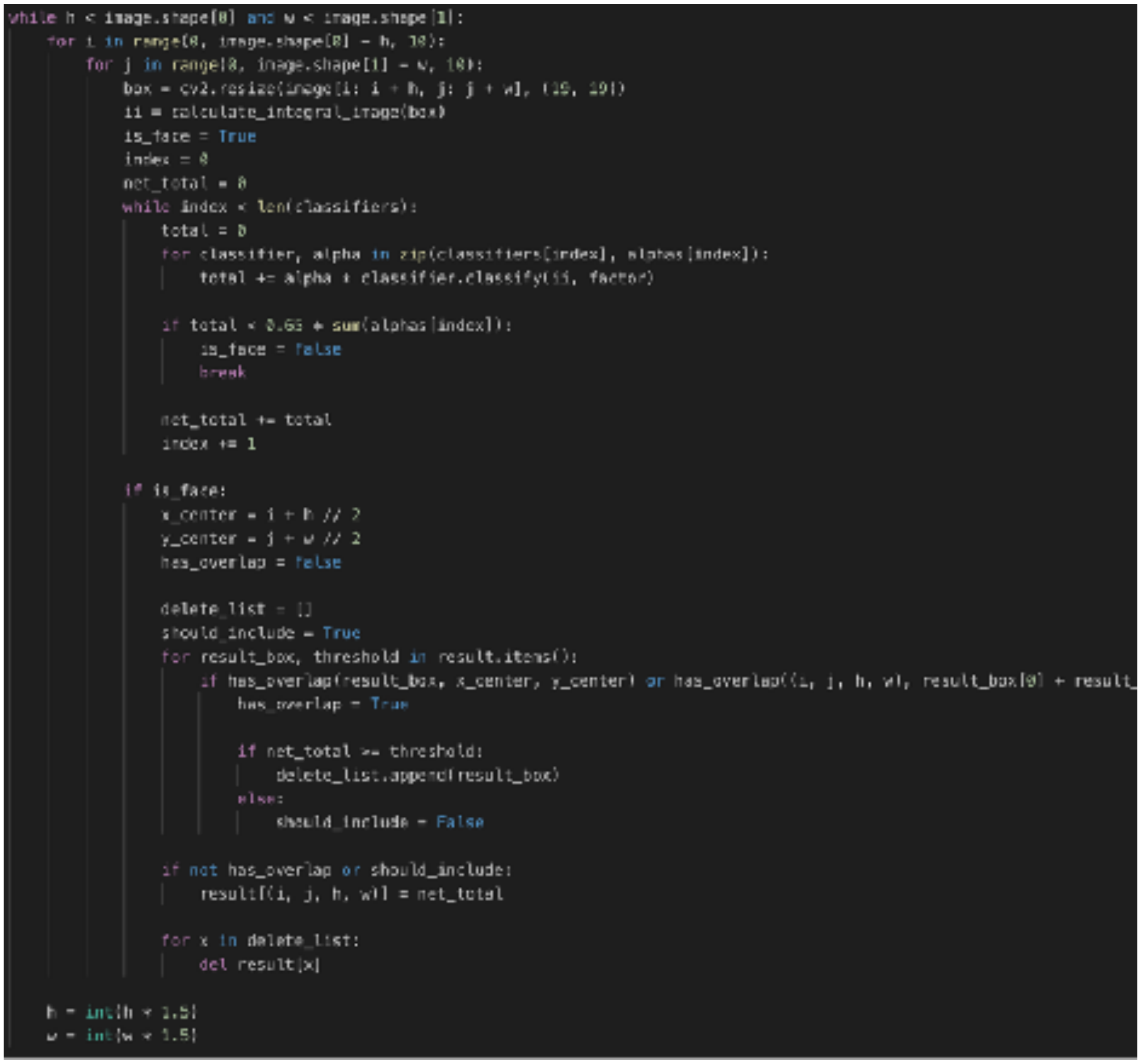


```python
class WeakClassifier:
    def __init__(self, feature, threshold, polarity):
        self.feature = feature
        self.threshold = threshold
        self.polarity = polarity

    def classify(self, integral_image, factor):
        positive_val = negative_val = 0
        for pos_feature in s compute_feature_value: compute_feature_value
            positive_val += compute_feature_value(pos_feature, integral_image)

        for neg_feature in self.feature[1]:
            negative_val += compute_feature_value(neg_feature, integral_image)

        feature_val = positive_val - negative_val

        return 1 if self.polarity * feature_val < self.polarity * self.threshold else 0
```

To test the classifier, we use the saved Models to evaluate the test images. To evaluate on image of big size, we use the sliding window approach. In this approach, we select a window of minimal size (100X100 in our case) from the image, resize it to 19X19 and classify it using the classifiers. The formula used for classification is shown below:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

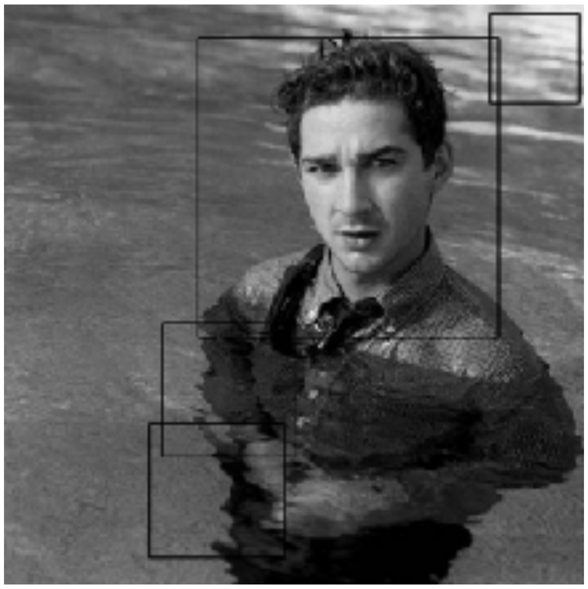
The classifiers return true if the sum of individual weak classifiers and alpha values is greater than 0.5 times sum of all alpha values from the classifier.
This resulted in a lot of bounding boxes around the facial regions of the image. To remove the unnecessary bounding boxes, we use a technique called “Non-maximal suppression”. In this technique we remove all bounding boxes which have more than 50% overlap with others and lesser score.

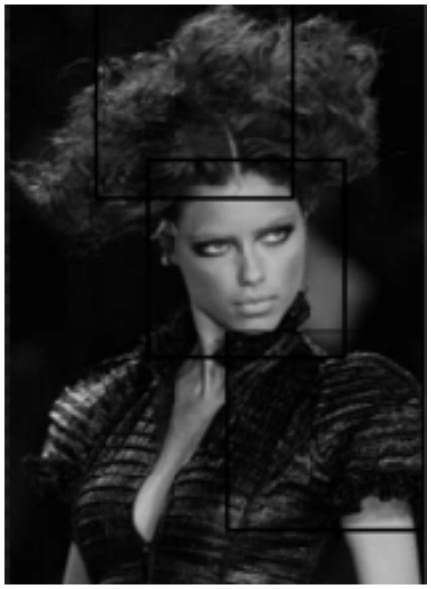


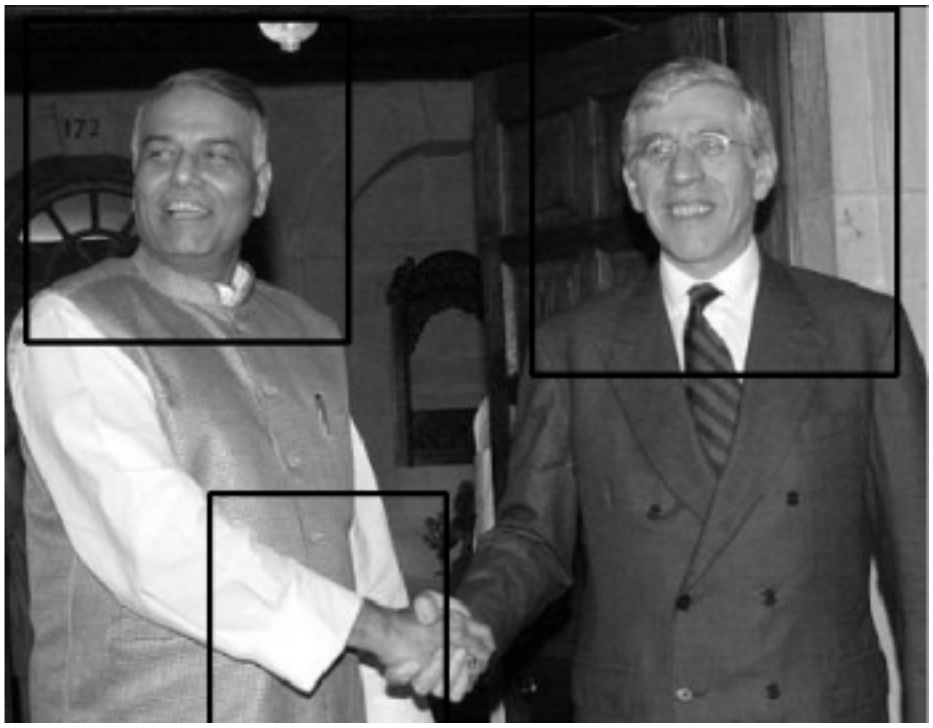
```
while h < image.shape[0] and w < image.shape[1]:
    for i in range(0, image.shape[0] - h, 10):
        for j in range(0, image.shape[1] - w, 10):
            box = cv2.resize(image[i: i + h, j: j + w], (19, 19))
            ii = calculate_integral_image(box)
            is_face = True
            index = 0
            net_total = 0
            while index < len(classifiers):
                total = 0
                for classifier, alpha in zip(classifiers[index], alphas[index]):
                    total += alpha * classifier.classify(ii, factor)

                if total < 0.65 * sum(alphas[index]):
                    is_face = False
                    break

                net_total += total
                index += 1

            if is_face:
                x_center = i + h // 2
                y_center = j + w // 2
                has_overlap = False

                delete_list = []
                should_include = True
                for result_box, threshold in result.items():
                    if has_overlap(result_box, x_center, y_center) or has_overlap((i, j, h, w), result_box[0] + result_
                        has_overlap = True

                        if net_total >= threshold:
                            delete_list.append(result_box)
                        else:
                            should_include = False

                if not has_overlap or should_include:
                    result[(i, j, h, w)] = net_total

                for x in delete_list:
                    del result[x]

    h = int(h * 1.5)
    w = int(w * 1.5)
```

Results:

The following results were obtained after running the trained classifier on a set of images:







Analysis:

The results of our classifier were pretty decent. Though it was able to recognize faces, it had some false positives and performed poorly in cases where the faces weren't straight (Side or tilted poses). This comes due to the fact that the training data was limited and so was the computation power and time required to train the classifier.

The following improvements are possible:

1) More training with huge datasets (that includes faces with different poses) which gives more accurate results.
2) More computation power and training time could significantly improve results.
3) Adding more cascade layers to improve test performance.

References and Citations:

1) https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
2) https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework
3) http://projects.i-ctm.eu/sites/default/files/PDF/1_admin/Tesi%20-%20Umar.pdf
4) https://medium.com/datadriveninvestor/understanding-and-implementing-the-viola-jones-image-classification-algorithm-85621f7fe20b
5) https://medium.com/datadriveninvestor/understanding-and-implementing-viola-jones-part-two-97ae164ee60f