SMAI Final Project Report

four_20
Siddik Ayyappa
P Balaramakrishna Varma
Miryala Narayana Reddy
Haasa Garikapati

Statistical Methods in AI

INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY
HYDERABAD

December 3, 2022

1 Introduction

The Project includes the implementation of the Viola Jones Algortithm. Viola Jones Algorithm is a very famous algorithm, for object detection at an extremely rapid rate. This work is distinguished by three key contributions.

a) Integral Image

b) Adaboost

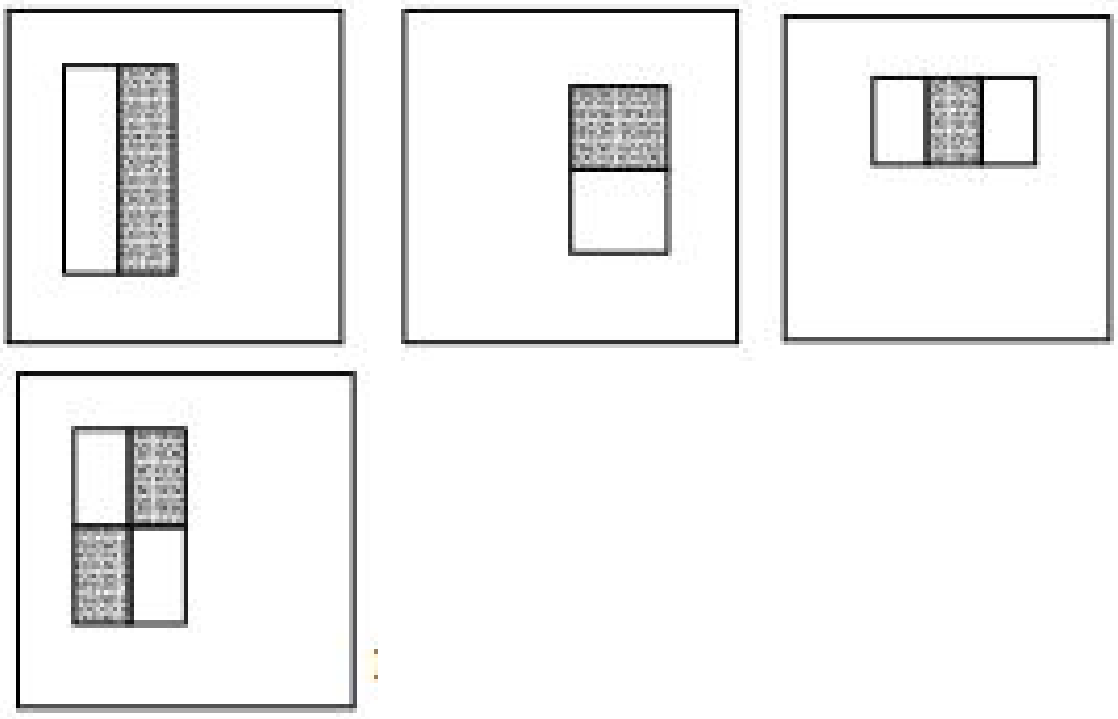
c) Cascade

In the domain of face detection the Viola Jones system yields detection rates comparable to the best previous systems. This paper brings together new algorithms and insights to construct a framework for robust and extremely rapid object detection. The integral image can be computed from an image using a few operations per pixel. Once computed, any one of these Harr-like features can be computed at any scale or location in constant time.

2 Feature Extraction

The motivation of using feature extraction is that it encodes raw ad-hoc pixel information which is not easy to learn while training. Also it has been observed that feature based networks work faster.
We have used following Rectangle Feature features.

- **Two Rectangle Features:** Difference between sum of pixels within two rectangle regions of same shape and size and are adjacent either horizontally or vertically.
- **Three rectangle feature**: Sum of pixels of two outside rectangles is subtracted from centre rectangle's sum of pixels.
- **Four rectangle feature**: Difference between the diagonal pairs



2.1 Integral Image

Since the feature extraction for a Image takes so long and computationally heavy, this paper came up with new concept called Integral Image which computes the above features very efficiently with less number of array accesses. **Integral** is an intermediate representation of image that helps us in calculating the rectangle features very rapidly and efficiently.
Integral Image at x, y is defined as sum of x', y' for all $x' \leq x$ and $y' \leq y$

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

the following recurrence relation can be used to compute the integral image in one pass

$$s(x, y) = s(x, y - 1) + i(x, y)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y)$$

with initial conditions $s(x, -1) = 0$ and $ii(-1, y) = 0$

2.2 Rectangular sum from integral images

We use this Integral Image to compute our Rectangular features using the following method. In the adjacent figure 1 we have integral images of all corners in A,B,C, D Rectangles. Sum of all pixels in D (rectangular sum) is

$$ii(x, y) + ii(x - 1, y - 1) - (ii(x - 1, y) + ii(x, y - 1))$$

which requires only 4 array references. So compute sum of pixels in a rectangle we need 4 references from integral image. Similarly, we see that Number of array references for

- Two rectangle feature is 6
- Three rectangle feature is 8
- Four rectangle feature is 9

This is used in computation of Rectangle features.

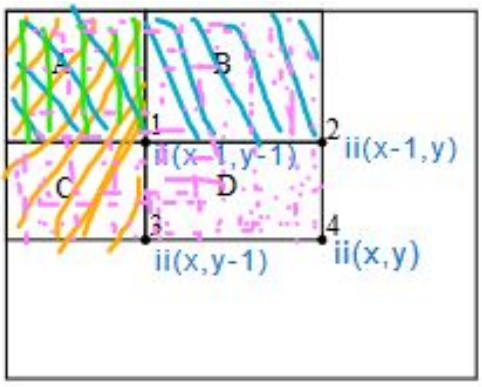


Figure 1: sum of pixels in rectangle D

2.3 optimization

Even computation of Integral Image features is taking long time. So we tried to optimize using multiprocessing on multiple CPUs and got $50\% - 60\%$ reduction in the time taken for computation on lower number of images. But when we use large number of images the reduction seems quite significant as it takes advantage of multiprocessing.

3 Classifier

Now that we have obtained the features and are provided with labels, we shall do classification with them. There are various machine learning techniques to do classification.The focus of the paper has mostly been on Adaboost Technique. As provided in the paper, we know that there have been over 180K features, associated with each sub-window of the image. Even if the features have a low computation time, computing all of the 180K features and using every one of them for classification is a computationally demanding task. Hence, we decide to look for the best features, to classify on and then make an effective classifier out of them.

In order to find the best features, we use a weak learning algorithm, which classifies on the basis of a single feature. (Weak classifiers, with Adaboost).And then we select the best features in each round for multiple rounds. The final classifier uses these (best) features. The pseudocode of the training process is given in the next slide. The final (strong) classifier is an ensemble of many weak classifiers.

3.1 Pseudo code:

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ 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 $t = 1, \ldots, T$:
  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, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  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 $\alpha_t = \log \frac{1}{\beta_t}$

3.2 Results:

The experiments in the paper have shown that using 200 features out of all the 180K+ features yields a 95% detection rate. (False Positive - 1/14084). Which is a good performance in 0.7 seconds. For better performance, we could try including more features in the classifier. However, this would increase the computation time. Including more features means that we have to do more rounds of boosting, which would lead us to the following conclusions. (Refer to Figure in the next page)

The Training time, would be directly increased, since we have to do multiple rounds of boosting depending upon the number of features we would like to include. The testing time, would be directly increased, since the image would be scanned by multiple classifiers. Even if the performance numbers are impressive, they are not impressive enough for real-time tasks.

3.3 Interpretation

The features obtained by the AdaBoost Algorithm are interpretable and meaningful. They focus on various regions of the face, like the nose bridge, eyes-nose-cheek regions etc. In the experiments conducted, it was seen that the first feature selected by AdaBoost focuses on the eye-nose-cheek regions. The second feature focuses on the nose-bridge regions.

4 Cascading classifier

The key insight to this method is that smaller, and therefore more efficient, boosted classifiers can be constructed which reject many of the negative sub-windows while detecting almost all positive instances. The threshold of a boosted classifier can be adjusted so that the false negative rate is close to zero.

Simpler classifiers are used to reject the majority of subwindows before more complex classifiers are called upon to achieve low false positive rates. A positive result from the first classifier triggers the evaluation of a second classifier, a positive result from the second classifier triggers the evaluation of third classifier and so on until we are satisfied with our false positive and detection rates.

Each stage in the cascade is constructed by training classifiers using AdaBoost and then adjusting the threshold to minimize false negative. The early cascade stages should be very simple and should eliminate as many negatives as possible while the further stages are more and more complex and take more time for evaluation. The examples which make it through the first stage are “harder” than typical examples and so on.

4.1 cascade training process

cascade training process involves two types of tradeoffs.

- classifiers with more features will achieve higher detection rates and lower false positive rates.
- At the same time classifiers with more features require more time to compute.

The number of classifier stages, the number of features in each stage, and the threshold of each stage, are traded off in order to minimize the expected number of evaluated features. Each stage is trained by adding features using modified Adaboost until the target detection and false positives rates are met. Stages are added until the overall target for false positive and detection rate is met.

4.2 Pseudo code

A simple framework for cascade training is given below:

- f = the maximum acceptable false positive rate per layer.
- d = the minimum acceptable detection rate per layer.
- Ftarget = target overall false positive rate.
- P = set of positive examples.
- N = set of negative examples.

```
F(0) = 1.0; D(0) = 1.0; i = 0

while F(i) > Ftarget
    increase i
    n(i) = 0; F(i)= F(i-1)

    while F(i) > f × F(i-1)
        increase n(i)
        use P and N to train a classifier with n(i) features using AdaBoost
        Evaluate current cascaded classifier on validation set to determine F(i) and D(i)
        decrease threshold for the ith classifier (i.e. how many weak classifiers need to accept for strong classifier to accept)
            until the current cascaded classifier has a detection rate of at least d × D(i-1) (this also affects F(i))
    N = ø
    if F(i) > Ftarget then
        evaluate the current cascaded detector on the set of non-face images
        and put any false detections into the set N.
```

5 Analysis

- All the results are gathered for 1000 training samples.
- We have trained a 5 layer cascade.
- The adaboost classifier performs similar to the results mentioned in the paper.

analysis

December 3, 2022

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from PIL import Image
from timeit import default_timer as timer

from classifier import *
from utils import *
from cascade import *
import features as fe
import utils as ut
import plotly.express as px
```

[2]:
```
X, y = Create_data(1000)
X_train, X_test, y_train, y_test = Split_Data(X, y)
```

```
number_of cpus =  16
```

0.1 Feature Analysis

We look the following plots - Time taken to convert image into integral immage vs size of the image. - No of feature vectors vs size of the image - Time taken to compute feature vectors vs size of the image. (sequential + parallel) - Time to compute i^{th} fixed vector for a fixed size of image. - Time to compute i^{th} feature vs size of the image.

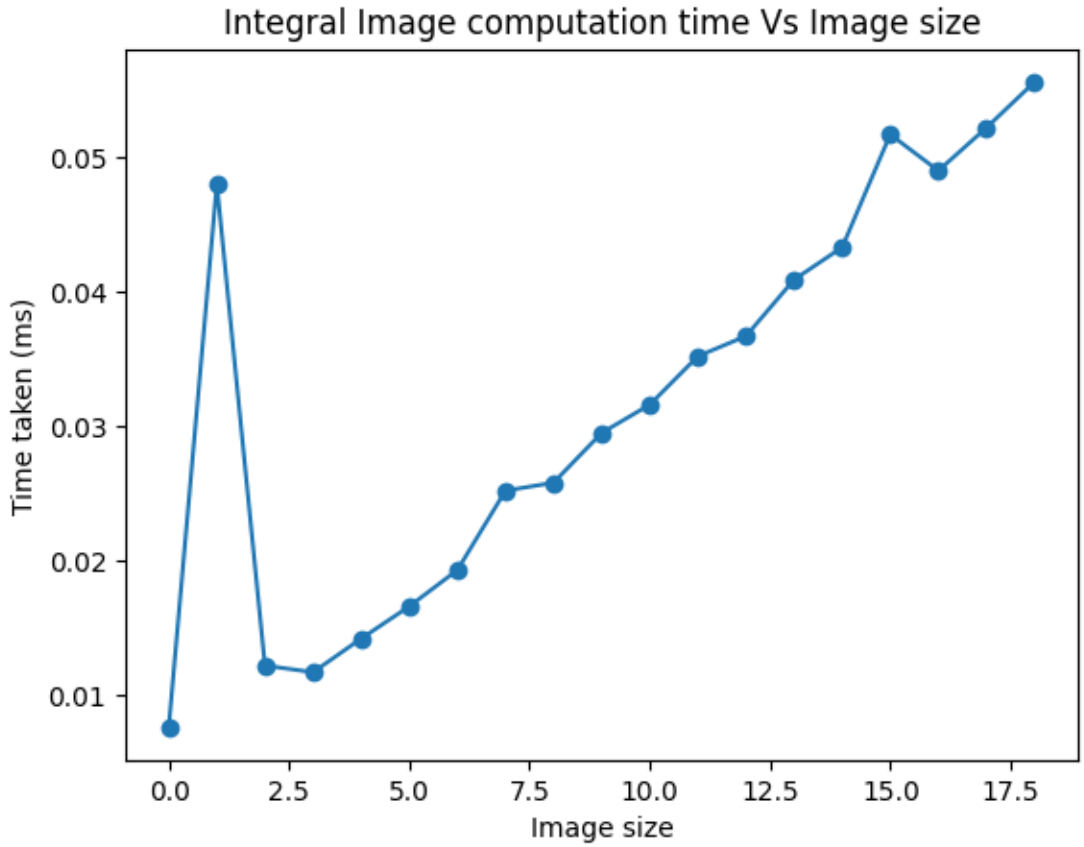
0.1.1 Time taken to convert image into integral image vs size of the image.

[3]:
```
image = Image.open("Data/faces/face.train/train/face/face00001.pgm")
image = np.array(image)
W,H = image.shape
print(W,H)
time = []
img_size = []
for i in range(W):
    img = image[:i,:i]
    s = timer()
    fe.caliculate_intergral_image(img)
```

```
    e = timer()
    time.append(1000*(e-s))
    img_size.append(i)

plt.plot(img_size,time,'o-')
plt.xlabel("Image size")
plt.ylabel("Time taken (ms)")
plt.title("Integral Image computation time Vs Image size")
plt.show()
```

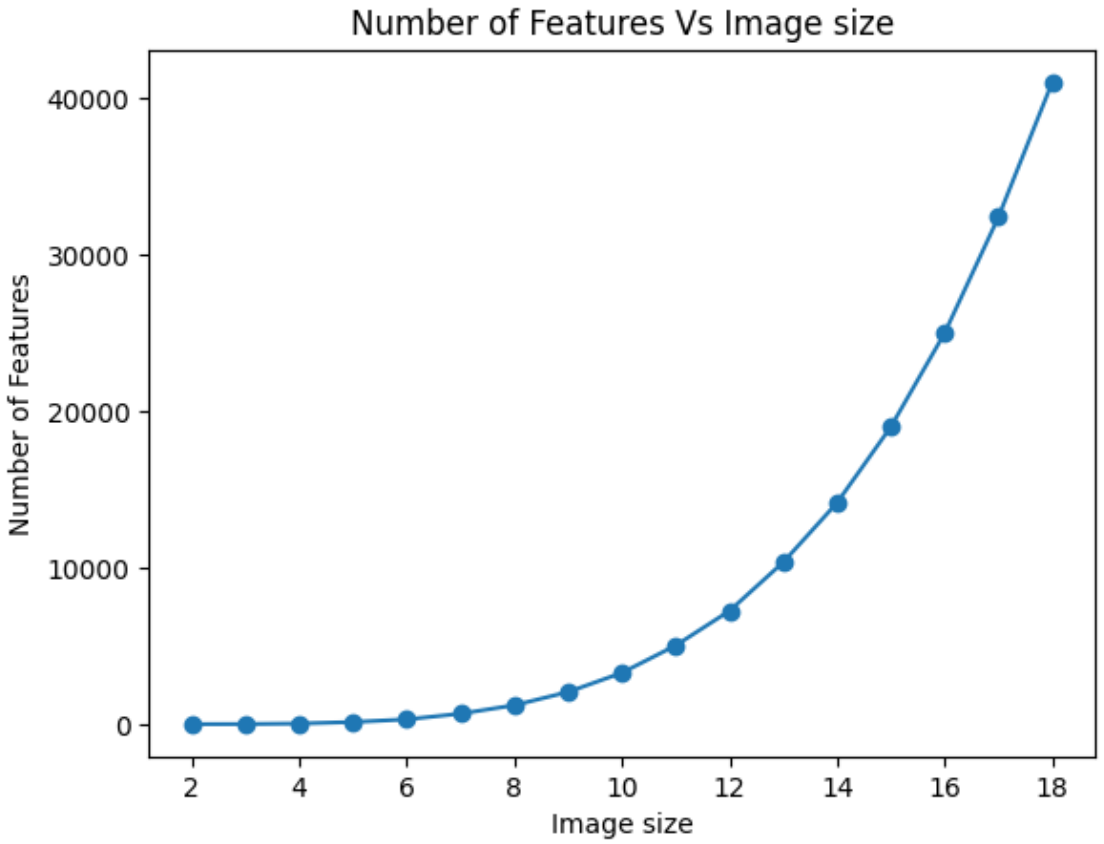
```
19 19
```



0.1.2 No of feature vectors vs size of the image

[4]:
```
image = Image.open("Data/faces/face.train/train/face/face00001.pgm")
image = np.array(image)
W,H = image.shape
f = []
img_size = []
for i in range(2,W,1):
```

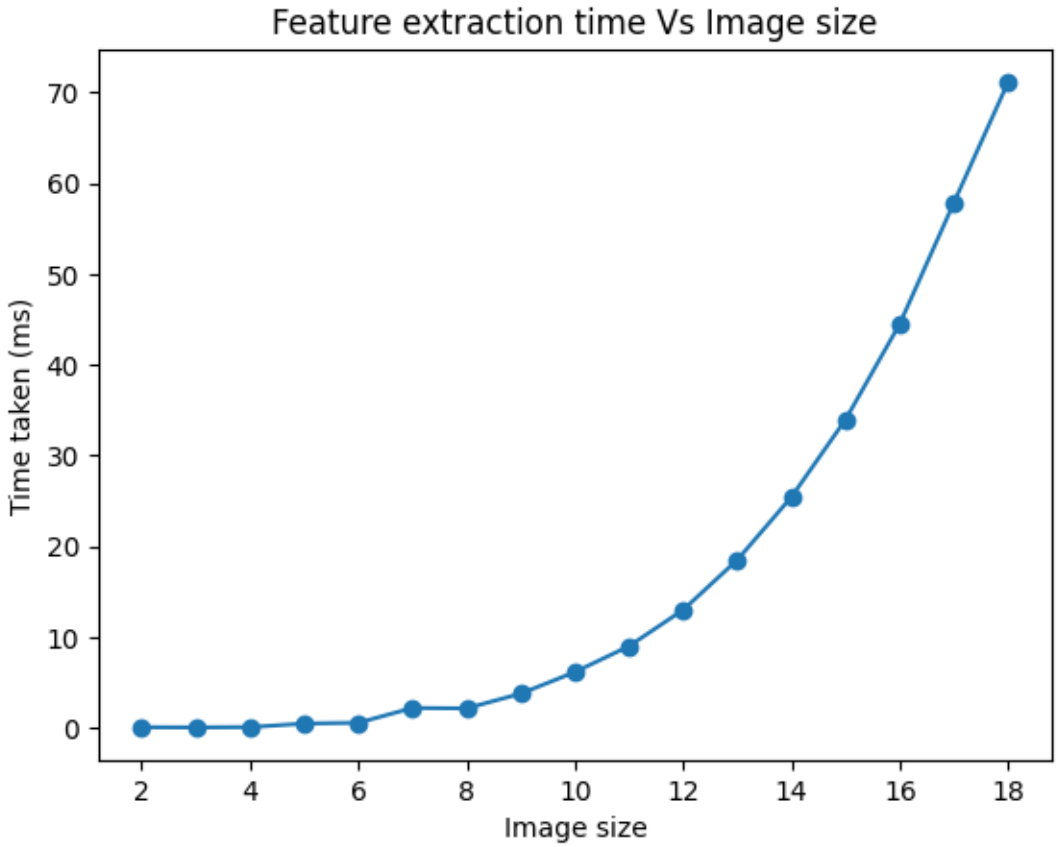
```
    img = image[:i,:i]
    rect = fe.get_rectanges(i,i)
    no_rect = fe.get_no_rectangles(i, i)
    nf = fe.feature_extraction_images(np.array([img]),rect,no_rect)
    img_size.append(i)
    f.append(nf.shape[1])
plt.plot(img_size,f,'o-',)
plt.xlabel("Image size")
plt.ylabel("Number of Features")
plt.title("Number of Features Vs Image size")
plt.show()
```



0.1.3 Time taken to compute feature vectors vs size of the image.

[5]:
```
t = []
img_size = []
for i in range(2,W,1):
    img = image[:i,:i]
    rect = fe.get_rectanges(i,i)
    no_rect = fe.get_no_rectangles(i, i)
```

```
    s = timer()
    nf = fe.feature_extraction_images(np.array([img]),rect,no_rect)
    e = timer()
    img_size.append(i)
    t.append(1000*(e-s))
plt.plot(img_size,t,'o-',)
plt.xlabel("Image size")
plt.ylabel("Time taken (ms)")
plt.title("Feature extraction time Vs Image size")
plt.show()
```



0.1.4 Time to compute i^{th} fixed vector for a fixed size of image.

[6]:
```
IntImage = fe.caliculate_intergral_image(image)
no_features = fe.get_no_rectangles(19, 19)
rectangles = fe.get_rectanges(19, 19)
times = []
for i in range(no_features):
    s = timer()
```

```
    for j in range(10):
        fe.get_nth_feature(IntImage, rectangles, i)
    e = timer()
    times.append(e - s)

# plot the time taken for each feature using plotly lin plot
fig = px.line(x=np.arange(no_features), y=times)
# details of the plot
fig.update_layout(
    title="Time taken for each feature",
    xaxis_title="Feature number",
    yaxis_title="Time taken (s)",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="#7f7f7f"
    )
)
fig.show()
```

0.1.5 Time to compute i^{th} feature vs size of the image.

[7]:
```
times = []
for size in range(3,19):
    int_image = fe.caliculate_intergral_image(image[:size,:size])
    rectangles = fe.get_rectanges(size, size)
    s = timer()
    for i in range(10000):
         fe.get_nth_feature(int_image, rectangles, 1)
    e = timer()
    times.append(e - s)

# plot the time taken for each feature using plotly line plot
fig = px.line(x=range(3,19), y=times)
#
fig.show()
```

0.2 Adaboost Analysis

We investigate the following plots - Accuracy vs number of features used in the adaboost classifier. - Positve Detection rate vs number of features used in the adaboost classifier. - False Positive rate vs number of features used in the adaboost classifier. - False Negative rate vs number of features used in the adaboost classifier - Time taken to classify subwindows vs number of features used in adaboost classifier.

[8]:
```
def details_vs_num_features(X_train, y_train, X_test, y_test):
    num_features = list(range(1, 50, 2))
```

```
    accuracy = []
    detection_rate = []
    false_positive_rate = []
    false_negative_rate = []
    clf = AdaBoostClassifier()
    for i in num_features:
        clf.fit(X_train, y_train, i)
        accuracy.append(clf.score(X_test, y_test))
        detection_rate.append(clf.detection_rate(X_test, y_test))
        false_positive_rate.append(clf.false_positive_rate(X_test, y_test))
        false_negative_rate.append(clf.false_negative_rate(X_test, y_test))
    return num_features, accuracy, detection_rate, false_positive_rate,␣
 ↪false_negative_rate

num_features, accuracy, detection_rate, false_positive_rate,false_negative_rate␣
 ↪= details_vs_num_features(X_train, y_train, X_test, y_test)
```

0.2.1 Accuracy vs number of features used in the adaboost classifier.

[9]:
```
fig = px.line(x=num_features, y=accuracy, title='Accuracy vs Number of␣
 ↪Features')
# change the names of the x-axis and y-axis
fig.update_xaxes(title_text='Number of Features')
fig.update_yaxes(title_text='Accuracy')
# adjust the size of the figure
fig.update_layout(
    autosize=False,
    width=700,
    height=500,
)
fig.show()
```

0.2.2 Positve Detection rate vs number of features used in the adaboost classifier.

[10]:
```
fig = px.line(x=num_features, y=detection_rate, title='Detection Rate vs Number␣
 ↪of Features')
# change the names of the x-axis and y-axis
fig.update_xaxes(title_text='Number of Features')
fig.update_yaxes(title_text='Detection Rate')
# adjust the size of the figure
fig.update_layout(
    autosize=False,
    width=700,
    height=500,
)
fig.show()
```

0.2.3 False Positive rate vs number of features used in the adaboost classifier.

[11]:
```
fig = px.line(x=num_features, y=false_positive_rate, title='False Positive Rate␣
 ↪vs Number of Features')
# change the names of the x-axis and y-axis
fig.update_xaxes(title_text='Number of Features')
fig.update_yaxes(title_text='False Positive Rate')
# adjust the size of the figure
fig.update_layout(
    autosize=False,
    width=700,
    height=500,
)
fig.show()
```

0.2.4 False Negative rate vs number of features used in the adaboost classifier

[12]:
```
fig = px.line(x=num_features, y=false_negative_rate, title='False Negative Rate␣
 ↪vs Number of Features')
# change the names of the x-axis and y-axis
fig.update_xaxes(title_text='Number of Features')
fig.update_yaxes(title_text='False Negative Rate')
# adjust the size of the figure
fig.update_layout(
    autosize=False,
    width=700,
    height=500,
)
fig.show()
```

0.2.5 Time taken to classify subwindows vs number of features used in adaboost classifier.

[18]:
```
X_train_fe, X_train, y_train = Create_data_imgs(200)
times = []
clf = AdaBoostClassifier()
for i in range(1,20):
    clf.fit(X_train_fe, y_train, i)
    s = timer()
    for index in range(10):
        clf.predict_img(X_train)
    e = timer()
    times.append(e - s)

# plot the time taken for each feature using plotly line plot
fig = px.line(x=range(1,20), y=times)
# details
```

```
fig.update_layout(
    title="Time taken for each feature",
    xaxis_title="Number of features",
    yaxis_title="Time taken (s)",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="#7f7f7f"
    )
)
fig.show()
```

```
number_of cpus =  16
```

0.3 Cascade Analysis

We investigate the following plots - number of training samples for each layer. - composition of training samples for each layer. - overall best accuraciy results. - Time taken to classify vs no of subwindows to be classified

[24]:
```
X_train_fe,y_train = Create_data(1000)
Strong_Classifiers,No_of_samples_per_layer,No_of_positive_samples_per_layer,No_of_negative_sam
 ↪= np.array(Train_Cascade(X_train_fe, y_train))
print("the length of the strong classifiers is", len(Strong_Classifiers))
```

```
number_of cpus =  16
the length of the strong classifiers is 5
```

0.3.1 number of training samples for each layer.

[25]:
```
# print the number of samples per layer vs the number of layers
fig = px.line(x=range(len(No_of_samples_per_layer)), y=No_of_samples_per_layer,␣
 ↪title='Number of Samples vs Number of Layers')
# change the names of the x-axis and y-axis
fig.update_xaxes(title_text='Number of Layers')
fig.update_yaxes(title_text='Number of Samples')
# adjust the size of the figure
fig.update_layout(
    autosize=False,
    width=700,
    height=500,
)
fig.show()
```

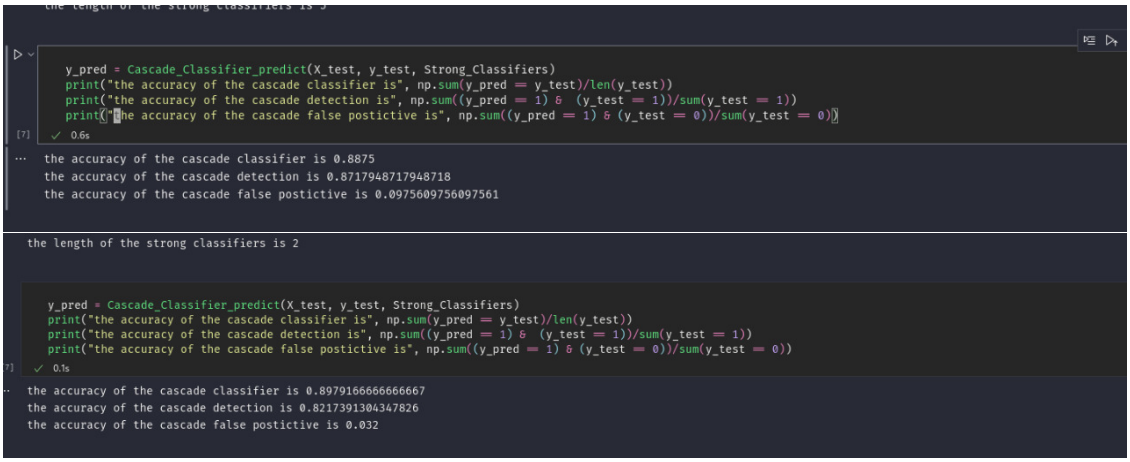
0.3.2 composition of training samples for each layer.

[26]:
```
# tabulate the number of positive and negative samples per layer
table = pd.DataFrame({'Number of Positive Samples':␣
 ↪No_of_positive_samples_per_layer, 'Number of Negative Samples':␣
 ↪No_of_negative_samples_per_layer})
table
```

[26]:

	Number of Positive Samples	Number of Negative Samples
0	431	369
1	325	106
2	314	11
3	314	0
4	314	0

0.3.3 Overall best accuracy results



0.3.4 Time taken to classify vs no of subwindows to be classified

[27]:
```
X_train, y_train = Create_data(200)
Strong_Classifiers, No_of_samples_per_layer, No_of_positive_samples_per_layer,␣
 ↪No_of_negative_samples_per_layer = np.array(Train_Cascade(X_train, y_train))

X_test_fe, X_test, y_test = Create_data_imgs(1000)
times = []
for i in range(0, 1000, 10):
    s = timer()
    Cascade_Classifier_predict_Img(X_test[:i], y_test[:i], Strong_Classifiers)
    t = timer()
    times.append(t-s)

# plot the time taken to predict vs the number of images
fig = px.line(x=range(0, 1000, 10), y=times, title='Time Taken to Predict vs␣
 ↪Number of Images')
```

```
# change the names of the x-axis and y-axis
fig.update_xaxes(title_text='Number of Images')
fig.update_yaxes(title_text='Time Taken to Predict')
# adjust the size of the figure
fig.update_layout(
    autosize=False,
    width=700,
    height=500,
)
fig.show()
```

```
number_of cpus =  16
number_of cpus =  16
```