**Classifier Adaboost Untuk Deteksi Wajah Menggunakan Algoritma Viola Jones**

**Tujuan utama artikel:**

Tujuan utama artikel ini adalah untuk mendemonstrasikan sebuah proyek yang menggunakan kumpulan data pelatihan yang berisi gambar wajah dan non-wajah berlabel untuk melatih pengklasifikasi Adaboost yang mengklasifikasikan apakah gambar yang diberikan adalah wajah atau non-wajah dan menggunakan gambar uji untuk menguji keakuratan classifier yang dipelajari.

**Bahasa pemrograman dan package yang digunakan:**

* Pickle
* Numpy
* Scikit-learn
* Matplotlib
* Tqdm
* Pandas
* Pillow

**Kegiatan utama dilakukan secara berurutan dalam proyek:**

* **Mengunduh dan memproses dataset**

Tugas awal adalah menemukan kumpulan data yang tepat yang berisi gambar wajah dan non-wajah untuk proyek kami.

Untuk kelas positif (gambar wajah), kami menggunakan dataset “[*A Century of Portraits: A Visual Historical Record of American High School Yearbooks*](https://people.eecs.berkeley.edu/~shiry/projects/yearbooks/yearbooks.html)”. Dataset tersebut berisi 37.921 gambar potret yang menghadap ke depan, tetapi kami hanya menggunakan 2000 gambar wajah karena keterbatasan sumber daya komputasi.

Untuk kelas negatif (gambar non-wajah) kami telah menggunakan set data latar belakang Stanford dari <http://dags.stanford.edu/projects/scenedataset.html> yang berisi 715 gambar dari berbagai latar belakang yang bukan “wajah”. Untuk menyeimbangkan kumpulan datanya, kami menggunakan augmentasi data untuk meningkatkan ukuran kumpulan data kami menjadi 1430 gambar dengan memotong wilayah secara acak dari 715 gambar asli yang telah kami gunakan sebagai gambar non-wajah.

Kami telah mengubah ukuran semua gambar menjadi ukuran seragam 22x22 dan membagi seluruh kumpulan data kami menjadi set pelatihan yang berisi 1600 gambar wajah dan 1144 gambar non-wajah dan satu set pengujian berisi 400 gambar wajah dan 286 gambar non-wajah.

* **Konversi Grayscale:**

Penelitian ini melakukan deteksi wajah pada gambar yang diberikan, dalam melakukan deteksi wajah pada gambar menggunakan metode viola jones, informasi warna tidak dibutuhkan. Oleh karena itu, peneliti melakukan konversi gambar menjadi gambar skala abu-abu (*grayscale*). Konversi gambar dilakukan baik pada gambar wajah maupun gambar non-wajah. Selain karena informasi warna yang memang tidak dibutuhkan, proses ini juga diperlukan untuk mengurangi beban komputasi ketika melakukan pemrosesan data gambar.

Gambar wajah setelah konversi skala abu-abu.

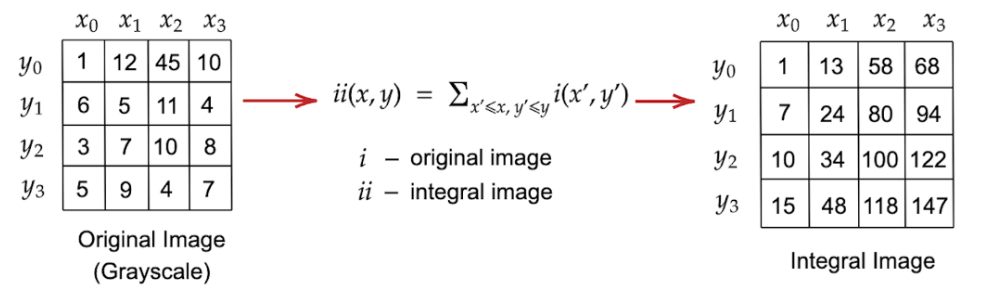
     

Gambar non-wajah setelah konversi skala abu-abu.

* **Melakukan Komputasi Gambar Integral:**

Pada tahap selanjutnya ialah melakukan perhitungan menggunakan gambar integral. Gambar integral digunakan agar setiap penjumlahan persegi panjang dapat dihitung dalam empat referensi larik. Fitur persegi panjang yang dihitung menggunakan gambar integral dapat dilakukan dengan lebih cepat. Citra integral yang berada pada lokasi (x,y) dihitung menggunakan jumlah piksel yang berada di atas dan di sebelah kiri x, y, inklusif. Gambar di bawah menggambarkan perhitungan gambar integral.

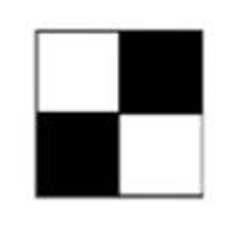
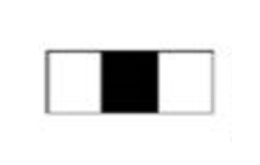
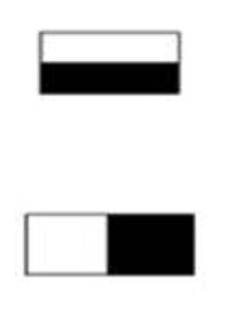


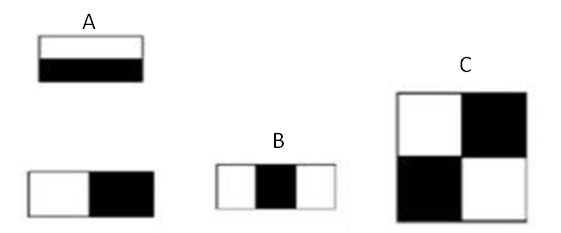
Menggunakan gambar Integral, jumlah persegi panjang apa pun dapat dihitung menggunakan empat referensi array. Karena proses mengekstraksi fitur haar-like melibatkan perhitungan jumlah persegi panjang dari wilayah yang lebih terang/gelap, pengenalan gambar Integral sangat mempercepat prosesnya. Dilakukan perhitungan gambar integral untuk seluruh kumpulan data kami. Kami kemudian melanjutkan untuk membangun fitur seperti haar-like.

* **Mengekstrak Fitur Haar:**

Viola dan Jones dalam makalahnya telah mendefinisikan 3 jenis fitur seperti Haar berikut ini sebagai berikut:

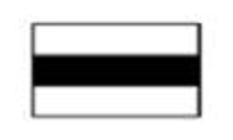
1. Edge feature



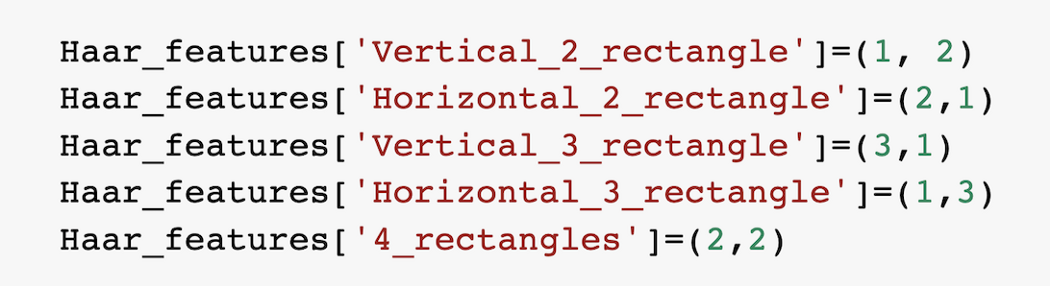
1. Line feature
2. Four-sided feature
3. 

Cara melakukan perhitungan pada fitur persegi panjang itu ialah dengan menjumlahkan piksel di dalam persegi panjang putih lalu dikurangi dengan jumlah piksel di dalam persegi panjang hitam. Gambar (A) menunjukkan edge feature. Gambar (B) menunjukkan line feature, dan (D) four-sided feature. Selain empat fitur di atas, peneliti juga mempertimbangkan fitur kelima dari fitur tipe garis dalam penelitian ini yang dapat dilihat pada gambar di bawah ini.

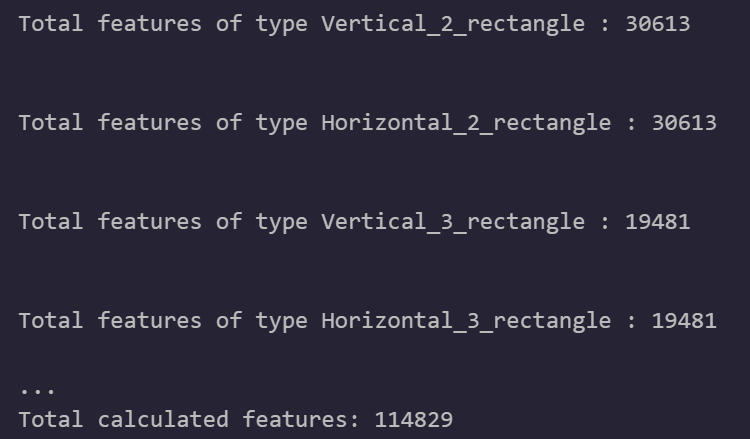
1. fifth line feature



Dalam kode, kami telah membuat *library* untuk lima jenis fitur haar-like di atas yang menyimpan tupel yang menunjukkan fitur Haar individual kami:

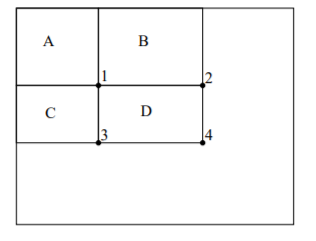


Jumlah total fitur dari tipe yang berbeda untuk gambar berukuran 22x22 ditunjukkan pada gambar di bawah ini:



Saat fitur ini dihamparkan pada gambar, nilai yang sesuai dengan wilayah terang ditambahkan dan nilai yang sesuai dengan wilayah gelap dikurangi dari penjumlahan di atas. Fitur haar-like ini membantu kami mengekstrak informasi berguna seperti tepi, garis lurus, dan diagonal yang dapat digunakan untuk mengidentifikasi objek, misalnya wajah manusia.

Bahkan untuk gambar kecil, kami akan mendapatkan banyak fitur haar, dalam kasus kami lebih dari 110.000 fitur untuk gambar 22 x 22. Untuk menghitung semua fitur secara efisien, Viola dan Jones memperkenalkan gambar integral yang telah kami sebutkan di langkah sebelumnya.



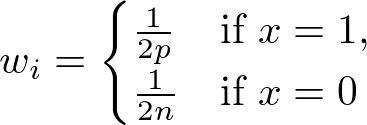
Menggunakan gambar Integral, jumlah piksel dalam persegi panjang D pada gambar di atas dapat dihitung dengan empat referensi larik. Jumlah piksel dalam persegi panjang A memberikan nilai gambar integral di lokasi 1 pada gambar di atas. A+B memberikan nilai di lokasi 2, A+C memberikan nilai di lokasi 3, dan A+B+C+D memberikan nilai di lokasi 4. Jumlah di dalam D dapat dihitung sebagai 4+1 — (2 +3). Dengan menggunakan metode ini kita dapat menghitung nilai dari semua fitur haar-like.

* **Algoritma Viola Jones**

Algoritme Viola-Jones menggunakan varian AdaBoost untuk pelatihan. Berikut ini langkah-langkah dari algoritma Viola-Jones:

1. Inisialisasi bobot untuk setiap contoh pelatihan

Di awal algoritma, kami memberi setiap contoh pelatihan bobot yang sama. Jika kelas positif berjumlah p dan kelas negatif berjumlah n, maka kami menetapkan bobot berikut untuk contoh pelatihan:



Di mana x adalah kelas dari contoh pelatihan.

1. Normalisasi semua bobot

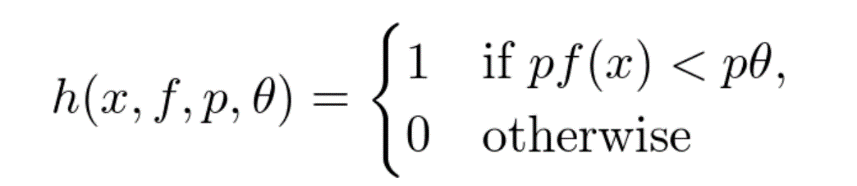
Fungsi pelatihan membutuhkan pemilihan *weak classifier* terbaik dan setiap fitur yang mungkin memberikan satu *weak classifier*. Jadi kami membangun semua fitur sebelum mengimplementasikan pelatihan dan menyimpan tipe setiap fitur (dari 5 tipe fitur haar-like yang ditentukan). Algoritma mengulang semua persegi panjang pada gambar dan memeriksa apakah mungkin membuat fitur.

1. Kemudian kami memilih *weak classifier* terbaik berdasarkan *weighted error* dari contoh pelatihan

Kami telah menerapkan fitur sebelum kami mulai melatih *classifier* karena nilai setiap fitur untuk gambar tidak pernah berubah. Menerapkan fitur sebelum pelatihan juga memungkinkan kami untuk memilih fitur sebelumnya (kami menggunakan SelectPercentile dari *library* sklearn.feature\_selection untuk memilih fitur terlebih dahulu) untuk mempercepat pelatihan.

1. Kemudian perbarui bobot sesuai dengan kesalahan dari *weak classifier* terbaik yang dipilih

Viola-Jones menggunakan serangkaian *weak classifier* dan menggabungkannya sesuai dengan bobotnya untuk mendapatkan *final strong classifier*. Setiap *weak classifier* melihat satu fitur ( f ). Setiap *weak classifier* memiliki ambang ( θ ) dan polaritas ( p ) untuk menentukan klasifikasi dari contoh pelatihan menurut persamaan berikut:



Setiap fitur adalah penjumlahan dari daerah persegi panjang positif dikurangi dengan jumlah daerah persegi panjang negatif.

Untuk setiap fitur, kami telah melatih satu *classifier* untuk menemukan ambang dan polaritas optimalnya menggunakan bobot contoh pelatihan. Setiap kali *weak classifier* baru dipilih sebagai yang terbaik, semua *weak classifier* harus dilatih ulang karena contoh pelatihan diberi bobot berbeda setelah setiap putaran. Karena ini adalah proses yang mahal secara komputasi, kami telah menggunakan metode yang dioptimalkan untuk melakukan pelatihan *weak classifier*. Pertama-tama mengurutkan bobot berdasarkan nilai fitur yang sesuai. Kemudian kami melakukan iterasi melalui rangkaian bobot dan menghitung kesalahan jika ambang batas dipilih untuk menjadi fitur tersebut. Kesalahan dihitung menggunakan persamaan berikut:

Error = minimum(P + TN – N, N + TP – P)

Di mana,

P = jumlah bobot dari semua contoh positif yang terlihat sejauh ini

N = jumlah bobot dari semua contoh negatif yang terlihat sejauh ini

TP = Jumlah total bobot semua contoh positif

TN = Jumlah total bobot semua contoh positif

Dengan menggunakan persamaan ini kita dapat menemukan *error* dari setiap *threshold* dalam waktu konstan dan *error* dari semua *threshold* dalam waktu linier. Kami menemukan ambang batas dan polaritas dengan kesalahan minimum. Nilai yang mungkin untuk ambang batas adalah nilai fitur pada setiap contoh pelatihan. Ambang batas diatur ke nilai fitur yang kesalahannya minimum. Polaritas ditentukan oleh berapa banyak contoh positif dan contoh negatif yang memiliki nilai fitur (dari fitur tertentu yang sedang dipertimbangkan) lebih rendah atau dan lebih besar dari ambang batas.

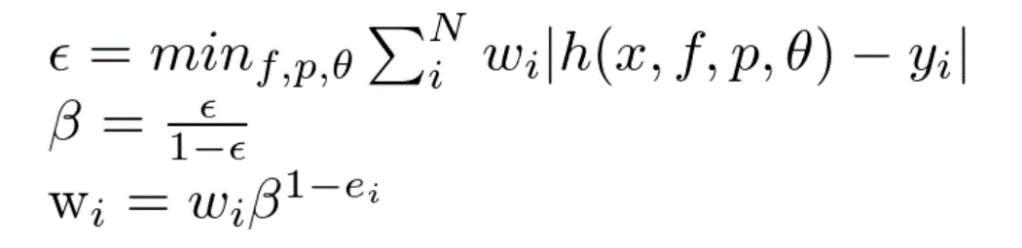
Polaritas p=1, Jika ada lebih banyak contoh positif dengan nilai fitur kurang dari ambang batas, selain itu p=-1.

1. Ulangi langkah 2–4 sebanyak N kali, di mana N adalah jumlah *weak classifier* yang diinginkan

Untuk memilih *weak classifier* terbaik di setiap putaran, kami melakukan iterasi melalui semua pengklasifikasi dan menghitung kesalahan *average weighted* dari setiap *classifier* pada dataset pelatihan, dan memilih *classifier* dengan kesalahan terendah.

1. Memperbarui bobot

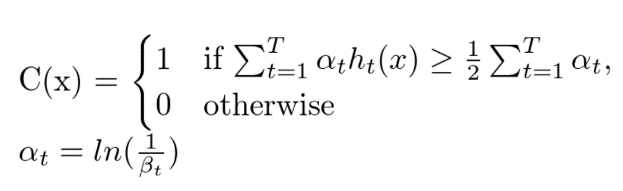
Setelah memilih *weak classifier* terbaik, kami memperbarui bobot dengan kesalahan dari *weak classifier* yang dipilih. Contoh pelatihan yang diklasifikasikan dengan benar diberi bobot yang lebih kecil, contoh yang diklasifikasikan dengan salah tidak memiliki perubahan bobotnya. Kami memperbarui bobot sesuai dengan persamaan berikut:



Wi adalah bobot dari contoh ke-i, 𝜖 diberikan oleh *error of weak classifier* terbaik, dan 𝜷 yang diperoleh dengan menggunakan persamaan di atas, memberikan faktor yang digunakan untuk mengubah bobot. Pangkat dari 𝜷 adalah 1-e di mana e adalah 0 jika contoh pelatihan diklasifikasikan dengan benar dan 1 jika diklasifikasikan dengan salah.

1. Strong Classifier

*Final classifier* didefinisikan sebagai berikut:



*Final classifier* adalah kombinasi linier berbobot dari T (dalam kasus kami T=10) jumlah pengklasifikasi lemah di mana bobot (dilambangkan dengan koefisien 𝛂) bergantung pada kesalahan karena merupakan log natural dari invers dari 𝜷. Jika jumlah tertimbang dari keputusan *weak classifier* lebih besar dari atau sama dengan setengah dari jumlah semua 𝛂, maka kami menetapkannya ke kelas 1 jika tidak, ia ditetapkan ke kelas 0.

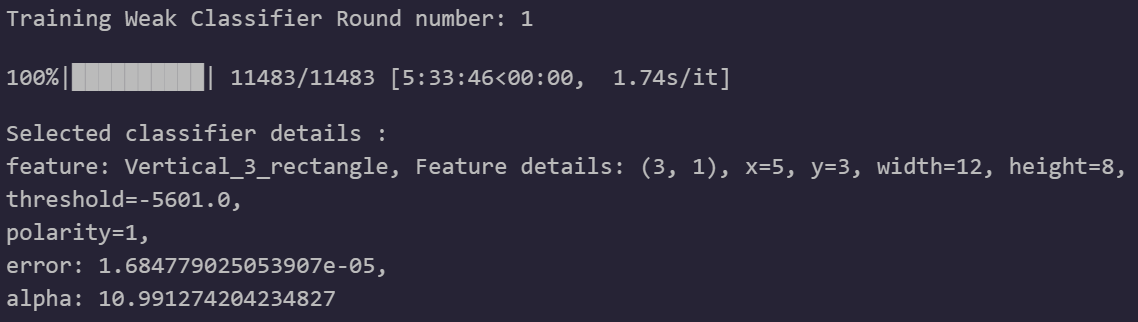
1. Attentional Cascade

*Attentional cascade* menggunakan serangkaian *classifier* Viola-Jones, masing-masing semakin kompleks untuk mengklasifikasikan gambar. *Classifier* digunakan dalam mode kaskade di mana hanya *classifier* pertama melatih semua contoh pelatihan, dan setiap *classifier* berikutnya dilatih pada semua contoh positif dan contoh negatif yang salah diklasifikasikan oleh *classifier* sebelumnya yaitu, false positif.

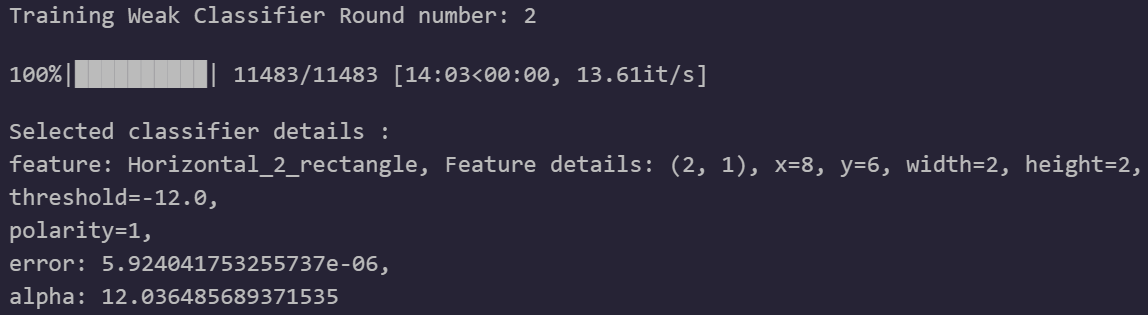
**Hasil Proyek**

Algoritma yang telah dibuat, selanjutnya peneliti jalankan sebanyak 10 putaran. Berikut adalah rincian pengklasifikasi lemah yang didapatkan oleh Adaboost pada setiap putaran yang telah dilakukan:

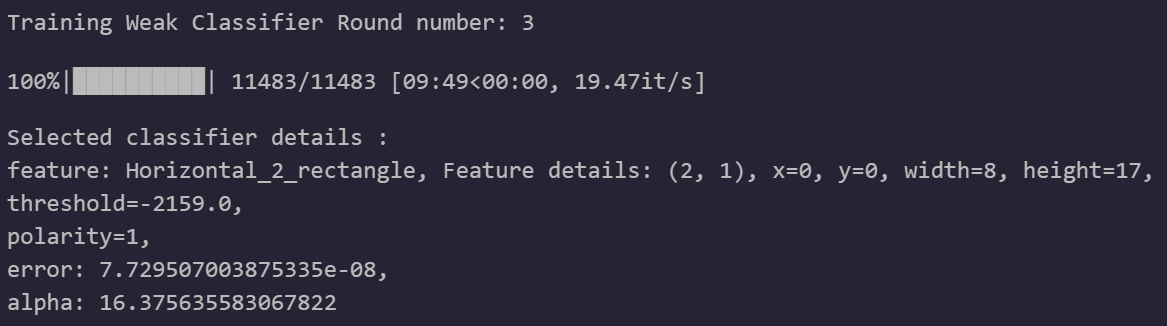
Putaran Pertama AdaBoost:



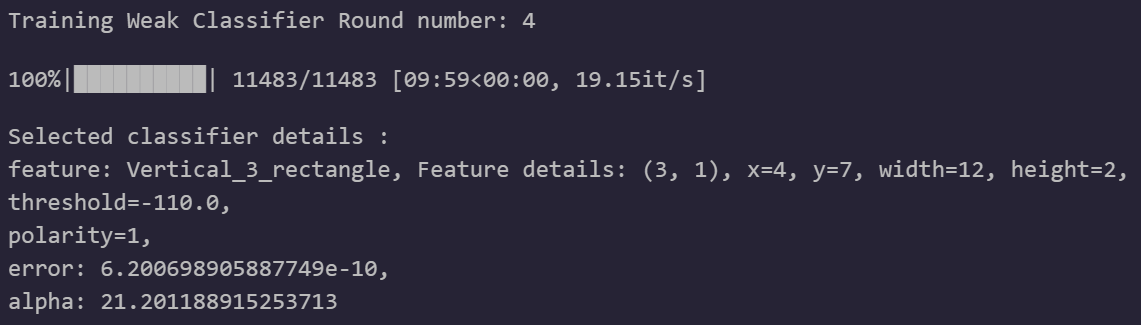
Putaran Kedua AdaBoost:



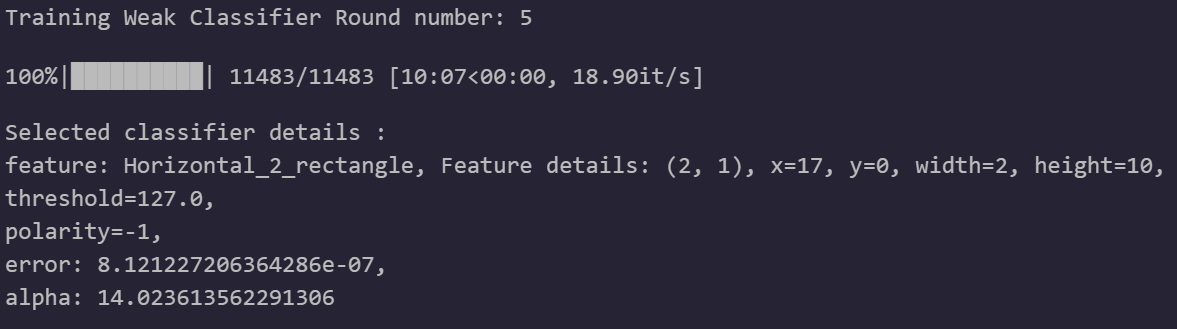
Putaran Ketiga AdaBoost:



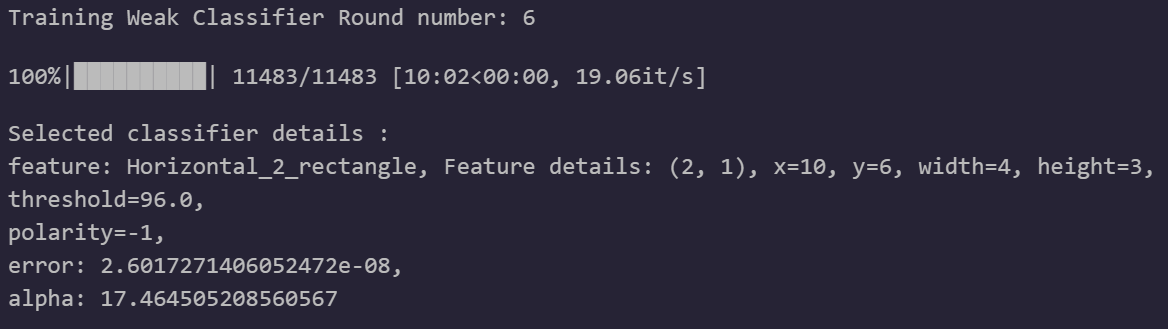
Putaran Keempat AdaBoost:



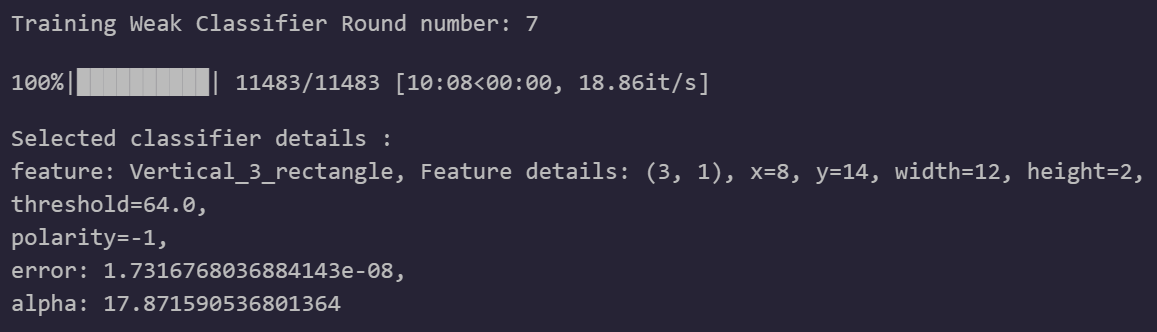
Putaran Kelima AdaBoost:



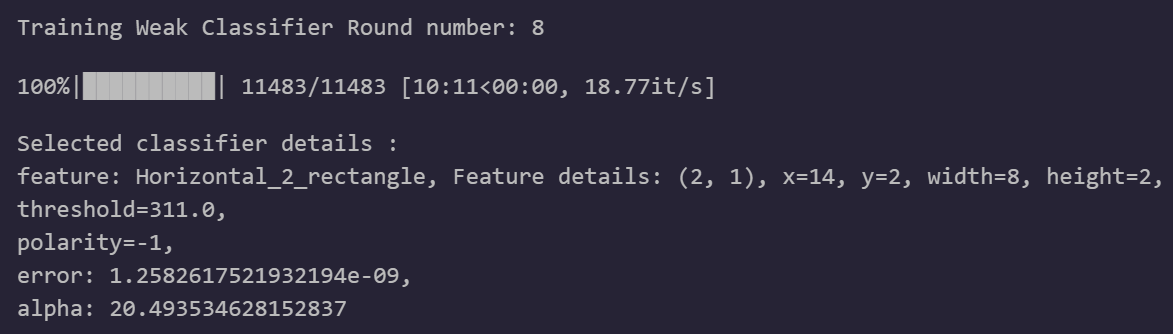
Putaran Keenam AdaBoost:



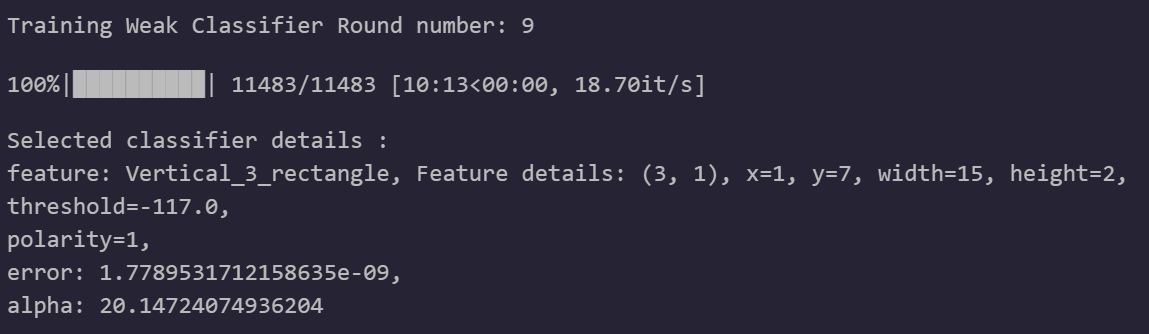
Putaran Ketujuh AdaBoost:



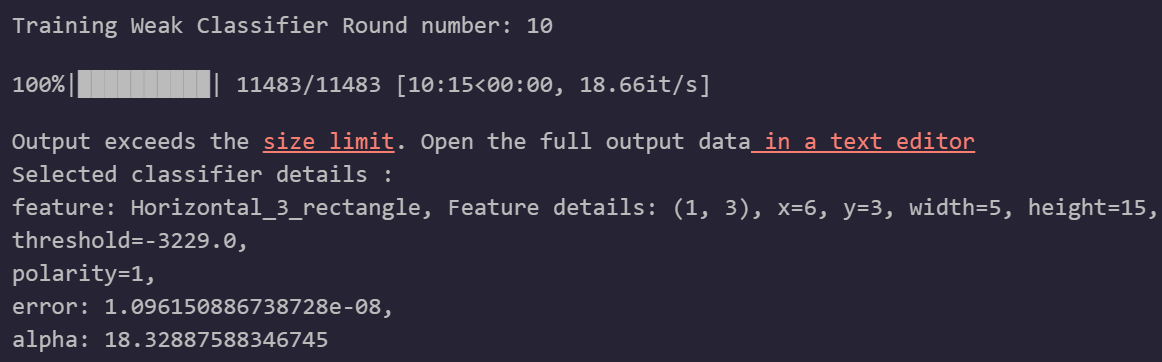
Putaran Kedelapan AdaBoost:



Putaran Kesembilan AdaBoost:



Putaran Kesepuluh AdaBoost:



Jadi inilah 10 fitur yang dipilih oleh Adaboost dalam 10 putaran seperti yang ditunjukkan di bawah ini:

* feature: Vertical\_3\_rectangle, Feature details: (3, 1), x=5, y=3, width=12, height=8
* feature: Horizontal\_2\_rectangle, Feature details: (2, 1), x=8, y=6, width=2, height=2
* feature: Horizontal\_2\_rectangle, Feature details: (2, 1), x=0, y=0, width=8, height=17
* feature: Vertical\_3\_rectangle, Feature details: (3, 1), x=4, y=7, width=12, height=2
* feature: Horizontal\_2\_rectangle, Feature details: (2, 1), x=17, y=0, width=2, height=10
* feature: Horizontal\_2\_rectangle, Feature details: (2, 1), x=10, y=6, width=4, height=3
* feature: Vertical\_3\_rectangle, Feature details: (3, 1), x=8, y=14, width=12, height=2
* feature: Horizontal\_2\_rectangle, Feature details: (2, 1), x=14, y=2, width=8, height=2
* feature: Vertical\_3\_rectangle, Feature details: (3, 1), x=1, y=7, width=15, height=2
* feature: Horizontal\_3\_rectangle, Feature details: (1, 3), x=6, y=3, width=5, height=15

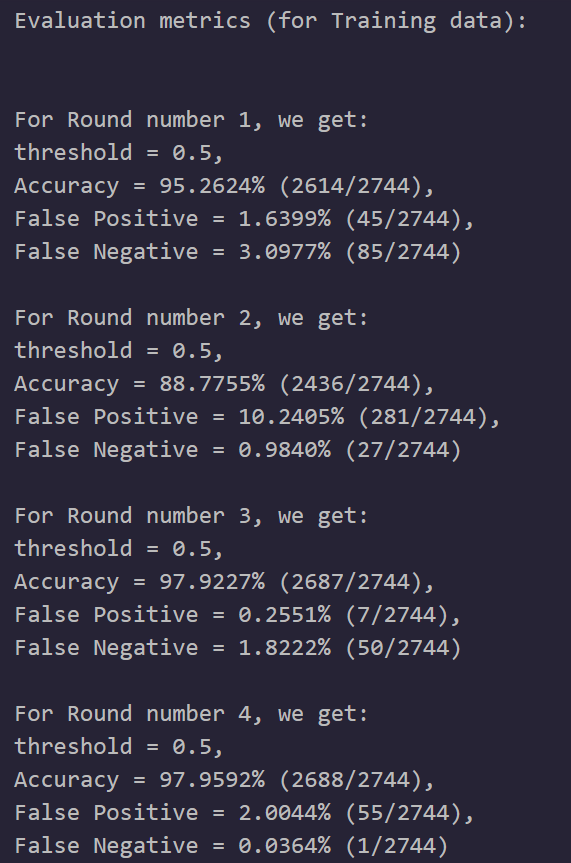
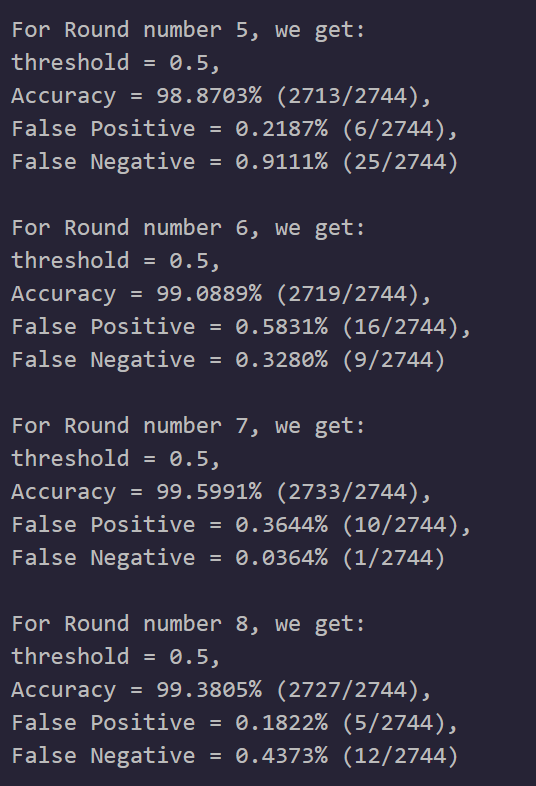
Gambar di bawah ini menunjukkan fitur teratas di setiap putaran:

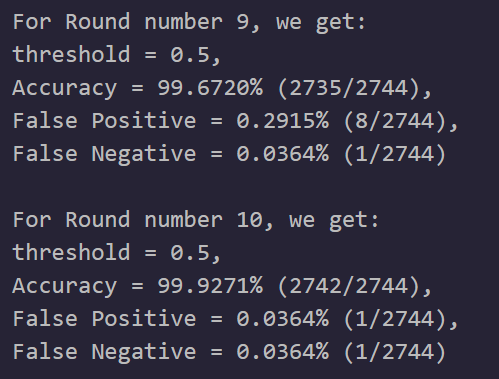
Dari gambar di atas, kita dapat melihat bahwa fitur Adaboost mencoba menangkap detail berikut dari wajah:

1. Detail mata (Putaran 5)
2. Detail bibir (Putaran 10 dan beberapa di Putaran 1)
3. Detail alis (Putaran 8)
4. Detail telinga (Putaran 9, Putaran 7)
5. Detail Dahi (Putaran 4)
6. Mungkin detail rambut dan telinga (Putaran 2)
7. Putaran 6 mencoba menangkap beberapa detail terkait mata dan hidung secara bersamaan.

Metrik evaluasi *classifier* pada dataset *training* kami mendapatkan tingkat akurasi setelah 10 putaran, yaitu 99,9271% dan tingkat akurasi maksimum yang diperoleh ialah pada putaran ke 10 dengan nilai 99,9271%.

Rincian metrik evaluasi classifier pada dataset training dapat dilihat pada tabel 1:

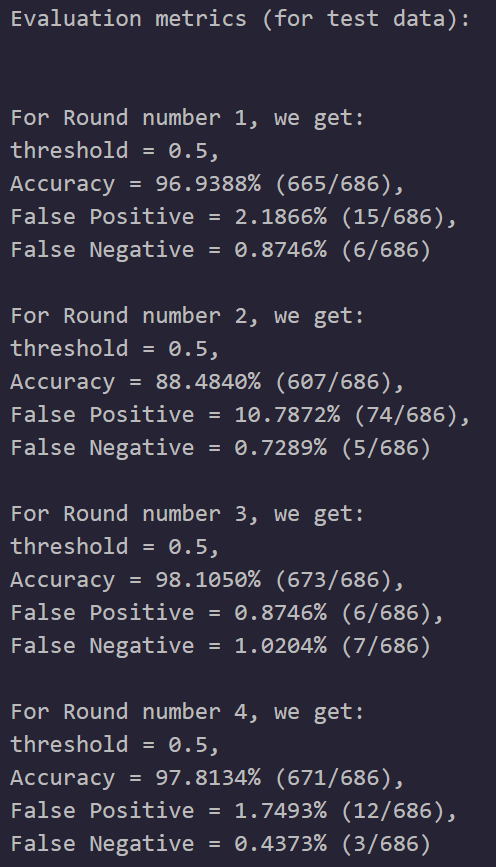
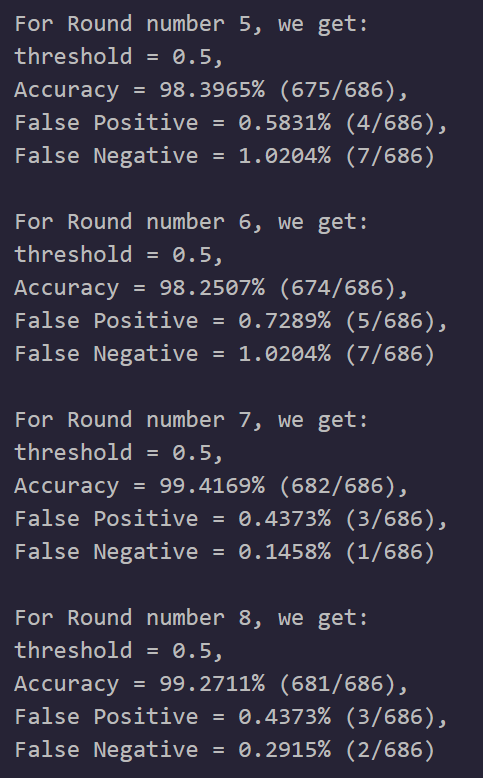
 

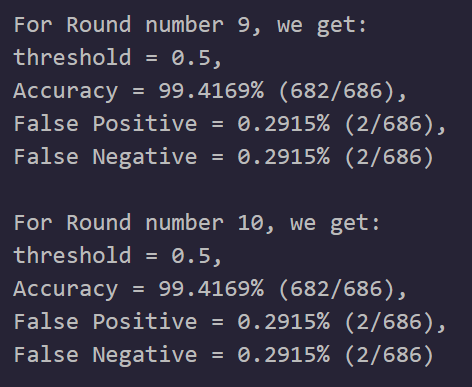


Metrik evaluasi *classifier* pada dataset *testing* kami:

Akurasi yang didapatkan pada dataset testing setelah 10 putaran adalah 99,4196%. Akurasi maksimum yang diperoleh adalah pada putaran ke 10 dengan nilai 99, 4196%.

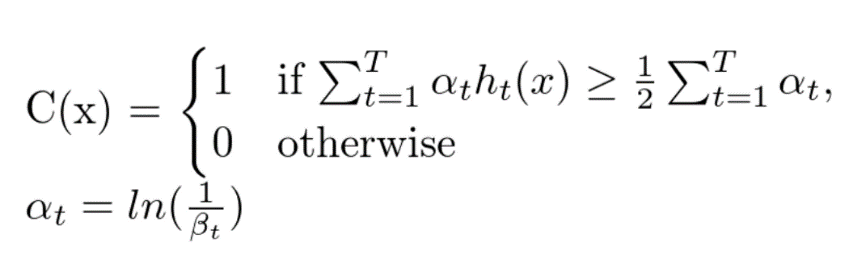
Berikut rincian metrik evaluasi:

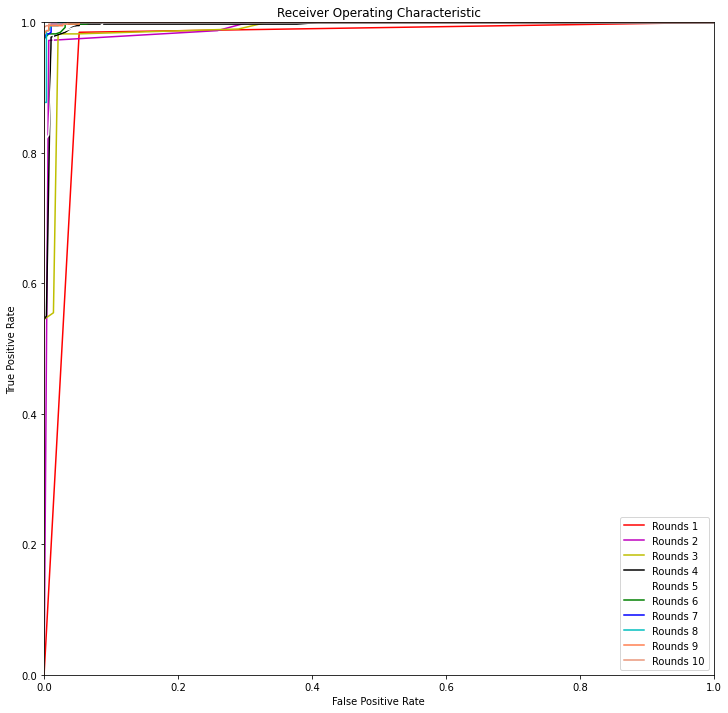


Kurva ROC dari *classifier* gabungan setelah menjalankan 1, 2, 3, 4, 5, 6, 7, 8, 9, dan 10 putaran peningkatan dan diterapkan pada dataset pengujian:

Kurva karakteristik operasi penerima, atau kurva ROC, adalah plot grafis yang mengilustrasikan kemampuan diagnostik sistem pengklasifikasi biner karena ambang diskriminasinya bervariasi.



Dari persamaan 8, kita dapat melihat bahwa threshold dari gabungan *classifier* diberikan oleh ½ \* sum(𝛂). Kami memvariasikan ambang dari *classifier* gabungan ini untuk menghitung Tingkat False Positif dan Tingkat True Positif untuk masing-masing ambang yang dipilih, yang kemudian kami plot untuk mendapatkan kurva ROC seperti pada gambar 19.



Dari kurva ROC yang ditunjukkan pada gambar 19, kita dapat melakukan pengamatan yaitu sebagai berikut:

Alasan *Classifier* di setiap putaran merupakan *classifier* yang baik dapat ditemukan pada beberapa hal berikut ini:

* Kurva classifier ROC memiliki Area under the curve (AUC) yang lebih besar daripada garis y=x yang merupakan garis yang melewati diagonal grafik. Garis diagonal y=x merepresentasikan random classifier yang secara acak mengklasifikasikan citra sebagai wajah dan non-wajah tanpa ada kemampuan untuk membedakan antara 2 kelas tersebut. Kita tahu bahwa semakin baik classifier, semakin besar AUC-nya. Jadi classifier kami bekerja lebih baik daripada classifier acak.
* Karena kurva ROC dari pengklasifikasi jauh dari garis diagonal, maka kita dapat mengatakan bahwa kita memiliki pengklasifikasi yang baik
* Kami memperoleh True Positive Rate (TPR) yang tinggi dengan False Positive rate (FPR) yang rendah untuk *classifier* kami yang menunjukkan bahwa kami memiliki *classifier* yang baik.

Alasan yang membuat classifier semakin baik saat putaran peningkatan meningkat, *dapat ditemukan pada beberapa hal berikut*:

* AUC kurva ROC meningkat dengan peningkatan putaran penguat yang menunjukkan bahwa kami memiliki *classifier* yang lebih baik
* Kami memperoleh True Positive Rate (TPR) yang lebih tinggi dengan False Positive Rate (FPR) yang lebih rendah dengan peningkatan putaran penguat yang menunjukkan bahwa kami memiliki *classifier* yang lebih baik
* Kurva ROC semakin jauh dari garis diagonal dengan setiap putaran peningkatan
* Dari grafik, kita dapat mengamati bahwa kurva ROC untuk Putaran 6, 7, 8, 9, 10 hampir tidak dapat dibedakan satu sama lain yang menunjukkan bahwa tidak banyak peningkatan kinerja pengklasifikasi setelah sejumlah putaran. Pengamatan ini dapat membantu kami mengetahui jumlah putaran peningkatan optimal yang diperlukan untuk melatih pengklasifikasi.

**Tantangan Yang Dihadapi Dan Langkah Yang Diambil Untuk Mengatasinya:**

1. Kami tidak dapat menemukan kumpulan data berlabel yang tepat yang berisi gambar wajah dan non-wajah secara bersamaan untuk tujuan pelatihan dan pengujian.

Langkah-langkah yang dilakukan untuk mengatasi:

Kami telah menggunakan gambar wajah dari satu sumber dan gambar non-wajah dari sumber yang berbeda dan digabungkan secara manual serta diberi label bersama untuk membuat kumpulan data pelatihan dan pengujian.

1. Kami memiliki batasan pada sumber daya komputasi untuk melatih model kami.

Langkah-langkah yang dilakukan untuk menghindari hal tersebut:

1. Kami mengubah ukuran semua gambar kami menjadi ukuran seragam 22x22.
2. Sumber data gambar wajah kami berisi 37.921 gambar potret wajah depan dan hanya kami gunakan sebanyak 2000 gambar wajah.
3. Kami menggunakan Google Colab untuk menjalankan seluruh kode kami yang memberi kami sumber daya komputasi yang lebih baik.
4. Sumber data gambar non-wajah kami hanya berisi 715 gambar yang menyebabkan ketidakseimbangan gambar kelas positif dan negatif dalam kumpulan data kami.

Langkah-langkah yang dilakukan untuk mengatasi:

Untuk menyeimbangkan kumpulan data kami, kami telah menggunakan augmentasi data untuk meningkatkan ukuran kumpulan data gambar non-wajah kami menjadi 1430 gambar dengan memotong wilayah secara acak dari 715 gambar asli.

1. Bahkan untuk gambar kecil, kami akan mendapatkan banyak fitur haar, dalam kasus kami lebih dari 110.000 fitur untuk gambar 22 x 22. Setiap putaran Adaboost membutuhkan banyak waktu untuk diselesaikan. Selain itu, banyak dari fitur ini tidak menawarkan banyak kekuatan klasifikasi dan sebagian besar tidak berguna.

Langkah-langkah yang dilakukan untuk mengatasi:

Kami telah menggunakan kelas SelectPercentile dalam paket SciKit-Learn yang memilih fitur berdasarkan persentil dari skor tertinggi, untuk mempersempit ruang fitur kami. Dengan menggunakan ini, kami telah memilih 10% teratas dari fitur potensial. Dengan ini, setiap putaran Adaboost membutuhkan waktu lebih sedikit untuk diselesaikan.