BAB II

KAJIAN PUSTAKA

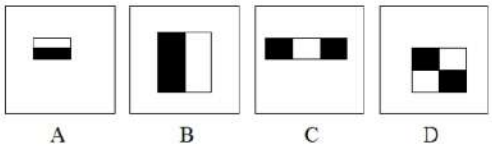
2.1 Pengertian Klasifikasi Objek

Fungsi dari klasifikasi objek adalah untuk memberikan deskripsi label ke sebuah segmentasi objek. Bila dilihat dari representasi fitur objek, ini dapat dicapai dengan melihat tanda-tanda keberadaan fitur yang mengindikasikan kelas dari objek. Hal ini umumnya dicapai dengan menentukan *threshold* diantara kelas-kelas yang direpresentasikan oleh *training set* yang sudah dilabeli. Pelatihan otomatis umumnya mementingkan distribusi dari setiap kelas (distribusi harus setara), dan melabeli setiap contoh dengan benar, Namun, isu yang umum didalam cara ini pemilihan fitur-fitur yang akan digunakan untuk klasifikasi objek kadang kurang sesuai dan sering kali bergantung terhadap keberuntungan untuk mendapat hasil sangat akurat (Renno dkk., 2007).

2.2 *Viola Jones Object Detection Framework*

Paul Viola dan Michael, J, Jones (ibid.) mempublikasikan sebuah makalah ilmiah dengan judul "*Robust Real-Time Face Detection*". Makalah tersebut mendeskripsikan sebuah framework pendeteksian wajah yang dapat memproses gambar secara cepat dengan tingkat akurasi yang tinggi. Ada tiga kontribusi penting dari makalah tersebut: Pertama adalah pengenalan sebuah representasi gambar baru yang dipanggil *Integral Image* yang memungkinkan penghitungan fitur yang dipakai oleh detektor dilakukan dengan cepat. Yang kedua adalah sebuah classifier yang efisien dan sederhana, yang dibuat menggunakan algoritma pembelajaran *Adaboost* (Freund dkk., 1996) untuk memilih sejumlah fitur-fitur kritis dari fitur-fitur potensial. Yang ketiga adalah sebuah metode untuk menggabungkan fitur-fitur tersebut dalam sebuah bentuk *cascade* yang memungkinakan algoritma untuk memfokuskan deteksi di area-area yang memiliki potensial saja.

2.2.1 *Features*



Gambar 2.1: Beberapa *Haar-like features* yang digunakan framework Viola-Jones.

Ada banyak alasan dimana penggunaan fitur lebih baik daripada nilai piksel secara langsung. Alasan paling umum adalah fitur dapat berfungsi untuk meng-*encode ad-hoc domain knowledge* yang sulit dipelajari dengan jumlah data pelatihan yang terbatas. Untuk sistem *framework Viola-Jones* ada alasan besar lainnya untuk penggunaan fitur yaitu sistem berbasis fitur beroperasi lebih cepat daripada sistem yang berbasis nilai piksel.

Fitur sederhana yang digunakan mirip dengan *Haar Basis Function* yang digunakan Papageorgiou dkk., 1998. Lebih tepatnya ada tiga jenis fitur: dua persegi panjang, tiga persegi dan empat persegi diagonal. Nilai dari sebuah *fitur dua persegi panjang* adalah perbedaaan diantara jumlah nilai piksel didalam dua atau lebih area persegi. Area-area tersebut memiliki ukuran dan bentuk yang sama, dan juga bersebelahan secara horizontal dan vertikal. Subuah *fitur tiga persegi* menghitung jumlah piksel dua area persegi di bagian luar dikurangi dengan jumlah nilai piksel persegi yang ada ditengah keduanya. Yang terakhir *fitur empat persegi* menghitung perbedaan nilai dari dua pasang persegi diagonal.

2.2.2 *Adaboost*

Dalam *framework Viola-Jones* sebuah varian dari *Adaboost* digunakan untuk memilih fitur dan juga untuk melatih *classifier*. Didalam bentuk aslinya, algoritma pembelajaran *Adaboost* digunakan untuk mem-*boost* performa klasifikasi dari algoritma pembelajaran sederhana. *Adaboost* melakukan ini dengan menggabungkan sekumpulan *classifier* lemah untuk membuat sebuah *classifier* kuat. Didalam istilah *Boosting*, *classifier* lemah disebut juga dengan *weak learner*. Sebagai contoh, algoritma pembelajaran *perceptron* mencari dari sekelompok *perceptron* dan mengambil *perceptron* dengan tingkat kesalahan klasifikasi terendah. Algoritma pembelajaran disebut lemah karena kita tidak berharap *classifier* terbaik untuk mengklasifikasi data dengan benar. Nyatanya, *perceptron* terbaik mungkin

Algorithm 1 SAMME Adaboost Algorithm

1: **Input:** Set latihan $\{(x_i, y_i)\}_{i=1}^{N}$ dimana x_i adalah sampel ke-i dan y_i adalah labelnya, jumlah *weak classifiers* M
2: **Output:** Urutan *classifier* $H(x)$
3: Inisiasi bobot gambar: $w_i^{(1)} = \frac{1}{N}, i = 1, 2, \ldots, N$
4: **for** $m = 1$ to M **do**
5: latih *weak classifier* $h_m(x)$ dengan distribusi $w_i^{(m)}$
6: Hitung error pada klasifikasi: $\epsilon_m = \sum_{i=1}^{N} w_i^{(m)} \cdot \mathrm{I}(h_m(x_i) \neq y_i)$
7: Hitung bobot *weak classifier*: $\alpha_m = \frac{1}{2} \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
8: Ubah bobot gambar sesuai error pada klasifikasi:
9: For $i = 1, 2, \ldots, N$: $w_i^{(m+1)} = w_i^{(m)} \cdot \exp(-\alpha_m \cdot \mathrm{I}(h_m(x_i) \neq y_i))$
10: Normalisasi bobot gambar: $w_i^{(m+1)} = \frac{w_i^{(m+1)}}{\sum_{i=1}^{N} w_i^{(m+1)}}$
11: **end for**
12: Hasil *strong classifier*: $H(x) = \mathrm{sign}\left(\sum_{m=1}^{M} \alpha_m \cdot h_m(x)\right)$

hanya memiliki tingkat akurasi 51%. Agar *weak learner* dapat di-*boost*, ia akan dipanggil untuk menyelesaikan sederet problem pembelajaran. Setelah satu ronde pembelajaran selesai, setiap contoh pembelajarannya akan dibobot ulang untuk menekankan problem yang salah diklasifikasi oleh *weak learner* sebelumnya. Bentuk *final strong classifier* adalah sebuah kombinasi *weak learner* berbobot. Untuk klasifikasi multi-kelas perlu dilakukan modifikasi kecil terhadap algoritma *boosting*. Hal ini dengan menambahkan perhitungan kecil pada penghitungan bobot, dengan memperhitungkan jumlah kelas yang akan diklasifikasi. (Hastie dkk., 2009)

2.2.3 *Weaklearn*

Framework Viola Jones menggunakan sebuah *weak learner* yang bernama *Decision Stump*, atau sebuah *Decision Tree* yang hanya memiliki dua daun kelas saja. *Decision Tree* sendiri mampu digunakan untuk permasalahan *multi-class*.

Decision Tree menghasilkan sebuah classifier didalam bentuk sebuah pohon pilihan, sebuah struktur yang berbentuk:

- Sebuah daun, mengindikasi sebuah kelas, atau
- Sebuah *decision node* yang menspesifikasi sebagian tes untuk dikerjakan atas sebuah nilai atribut, dengan satu *branch* dan *subtree* untuk setiap hasil dari tes.

Sebuah *Decision Tree* dapat digunakan untuk mengklasifikasi sebuah kasus

Algorithm 2 Metode Pembuatan *Decision Tree*

1: Anotasi semua dataset sesuai kelasnya. *Decision Tree* lalu akan dimulai dari akar
2: Pilih atribut terbaik untuk melakukan *split* dengan melakukan *Information Gain* (IG):

$$IG(S, A) = Entropi(S) - sum_v \frac{S_v}{S} x Entropi(S_v) \tag{2.1}$$

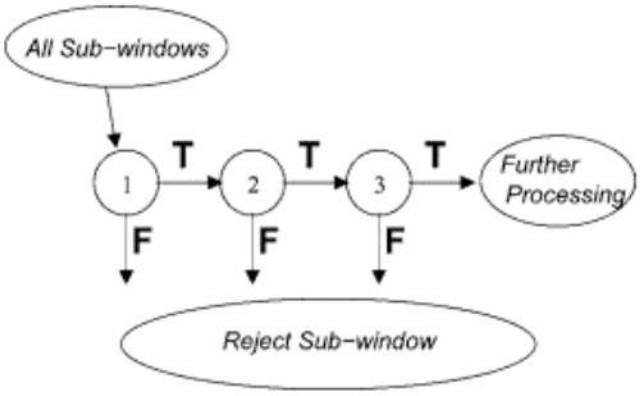
dimana:

- S: kumpulan sampel pada *node* sekarang
- A: atribut yang digunakan untuk *split*
- v: Sebuah nilai atribut A yang membagi S menjadi turunan S_v
- Entropi(S) = $-sum_c p_c log_2(p_c)$: Ukuran kemurnian pada kumpulan contoh dengan target label. Dimana p_c adalah proporsi contoh S yang ada pada kelas c.

3: *split* pohon ke *node* turunan sesuai atribut yang dipilih
4: tentukan apabila seluruh contoh sudah jatuh ke kelas yang benar((information gain) = 0) atau tidak bila tinggi maksimum sudah dicapai, bila tidak maka ulangi langkah 2 dan 3.

dengan memulai tes dari akar pohon dan bergerak sampai sebuah daun kelas ditemukan. Pada setiap *node* yang bukan merupakan daun, hasil dari tes kasus dideterminasi dan tes dilanjutkan ke *node* berikutnya sesuai dengan hasil tes tersebut. Hingga proses pada akhirnya (dan dengan pasti) menuju ke sebuah daun kelas.

2.2.4 *Attentional Cascade*



Gambar 2.2: *Workflow* dari *Attentional Cascade*

Attentional Cascade adalah sebuah *cascade* dari banyak *wea kclassifier* yang dibuat untuk meningkatkan performa deteksi dengan secara radikal mengurangi jumlah komputasi. Intinya *weak classifier* yang telah di-*boost* dapat dibuat lebih kecil dan efisien, yang dapat menolak mayoritas *sub-window* negatif dan mendeteksi sebagian besar dari *sub-window* positif. *Classifier* yang lebih sederhana digunakan untuk menolak mayoritas *sub-window* sebelum *classifier* yang lebih kompleks dipanggil untuk menurunkan tingkat *false positives*.

Struktur dari *cascade* merefleksikan fakta bahwa pada gambar apapun mayoritas *sub-window* pasti negatif. Oleh karena itu, *cascade* berusaha untuk sebanyaknya menolak sub-window negatif pada tahapan seawal mungkin. Sementara hasil positif akan memicu evaluasi dari setiap *classifier* dalam *cascade*, hal ini sangatlah langka.

Layaknya sebuah *Decision Tree*, sebuah *classifier* dilatih menggunakan contoh-contoh yang telah berhasil melewati tahap sebelumnya. Oleh karenanya, *classifier* pada tahap kedua menghadapi tantangan yang jauh lebih sulit daripada yang pertama.

Cascade dimulai dengan membuat sebuah *stage* awal, dimana *weak classifier* ditambahkan secara perlahan hingga *detection rate* yang diinginkan tercapai. *Detection rate* ini harus disesuaikan oleh pengguna sesuai dengan keperluannya, tujuannya tetap untuk mengurangi waktu komputasi tapi juga mencoba agar tidak terlalu banyak kasus *false positive* yang dapat lewat.