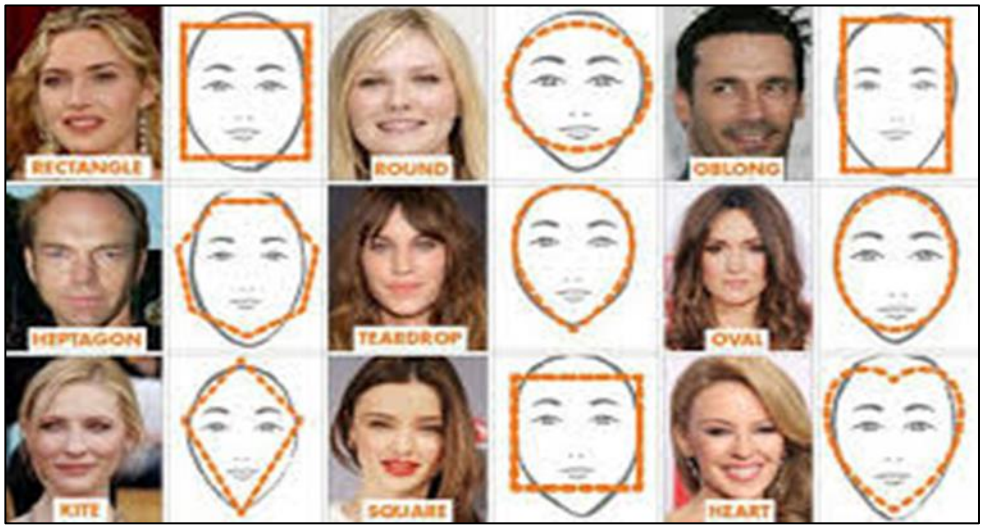
BAB 2

TINJAUAN PUSTAKA

2.1. Bentuk Wajah Manusia

Bentuk wajah dari tiap individu umumnya akan berbeda, dengan pengecualian individu yang memiliki kasus kembar identik. Penelitian dan survei yang dilakukan oleh Schmid(2020) menyatakan bahwa ada 5 bentuk wajah yaitu persegi, lingkaran, oval, hati, dan persegi panjang / oblong, dan beberapa bentuk wajah baru yang kurang umum seperti air mata, layangan, persegi panjang, dan heptagon. Gambar 2.1. berisikan bentuk bentuk wajah yang ada.



Gambar 2. 1. Contoh bentuk wajah (dailymail.co.uk, 2014)

1) Persegi

Bentuk muka yang lumayan simetris. Kening dan dagu memiliki panjang yang hampir sama, begitu pula dengan tulang pipi, kening, dan rahang yang seimbang lebarnya

2) Bundar

Memiliki fitur wajah yang seimbang, dengan panjang wajah yang lebih pendek dan pipi dan rahang yang lebih kecil

3) Hati

Memiliki kening yang lebih lebar dan panjang, dengan tulang pipi yang penjangnya juga hampir sama. Mengecil di area dagu

4) Oval

Secara umum lebih kecil dari bentuk muka lain, memiliki tulang pipi yang lebar dan menyempit di kening dan rahang.

Bentuk selanjutnya adalah bentuk – bentuk yang ditemukan oleh Schmid (2014) pada penelitiannya di Australia.

5) Persegi Panjang / Oblong

Rahang yang tegas dengan lebar yang relative sama dengan tulang pipi dan kening,

6) Air Mata

Ukuran muka yang lebih pendek dengan kening dan tulanng pipi yang sempit. Bentuk ini punya panjang kening yang lebih pendek dari dagu

7) Layangan / permata

Dicirikan dengan tulang pipi yang tegas, dan sedikit lebih lebar dari kening. Menyempit di area rahang dan dagu

8) Heptagon

Umumnya lebih panjang dari bentuk muka lain. Dilengkapi dengan kening yang lebar dan tulang pipi yang tegas

2.2. Face Recognition

Menurut Wilmer (2017) *face recognition* merupakan sebuah kemampuan untuk mengingat identitas dari berbagai wajah yang baru, dan lalu dapat memilih, mengembangkan, dan membedakannya dari wajah – wajah lainnya. Pada tahun 1960, Woodrow W. Bledsoe mengembangkan sebuah sistem yang dapat mengklasifikasi foto menggunakan RAND tablet. Menggunakan alat tersebut, koordinat akan diinput pada sebuah grid sacara manual dengan sebuah stylus. Sistem tersebut nantinya merekam fitur wajah seseorang ke dalam database dan nantinya akan dibandingkan dengan gambar lainnya (West, 2017). Kedepannya, sistem tersebut akan mendasari majunya sistem *face recognition*.

Dalam dunia pemrograman, algoritma *face recognition* merupakan sebuah program yang dapat mengetahui dan mengenali gambar wajah seseorang. Program ini membutuhkan pelatihan menggunakan *machine learning* untuk mendapatkan data wajah yang telah dipelajari untuk dibandingan, lalu hasil perbandingan akan disimpan di database (Geitgey, 2018). Implementasi algoritma ini dapat dilihat pada sistem tag milik Facebook.

Program *face recognition* dapat dibangun menggunakan beberapa algoritma pendukung. Algoritma – algoritma seperti PCA, OpenFace, OpenCV, Local Binary Pattern Histogram (LBPH) dan deskriptor HoG dapat digunakan sebagai algoritma pendukung dalam membangun program *face recognition* (doPrado dan Rosebrock, 2017).

2.3. Histogram of Gradients

Histogram of Gradients atau HoG merupakan salah satu tipe deskriptor yang bekerja untuk mendeteksi objek pada suatu gambar. Gambar yang akan dibandingkan akan di pre-proses terlebih dahulu sesuai kebutuhan. Angkah terakhir yaitu deskriptornya akan menghitung gradiennya kemudian kontur dan siluet gambar tersebut akan diambil dan diperlihatkan. Mallick (2016) menjelaskan tentang cara kerja HoG yang seperti berikut.

Pertama – tama yang dilakukan yaitu mempreproses gambar yang akan digunakan ke dalam bentuk gradiennya. Umunya gambar akan dikonversi ke bentuk *grayscale.* Dalam proses ini, tiap pixel akan memiliki nilai gradient dan juga magnitude dan arah yang didefinisikan menggunakan rumus :

$$g = \sqrt{g^2x + g^2y} \tag{2.1}$$

$$\theta = arctan\, {}^{gy}/_{gx} \tag{2.2}$$

Dengan g = arah gradient

Dan θ = magnitude

Arah tersebut nantinya dimasukkan ke dalam bin berukuran 9 angka yang sesuai dengan arah gradient yang dihitung pada sudut 0 – 180 derajat dan disebut unsigned gradient. Karena itu perhitungan arah dibagi tiap 20 derajat (0, 20, …, 160)

Berikutnya gambar akan dibagi ke dalam sel berukuran 8 x 8. Tiap sel dihitung besaran arah dan magnitudenya. Penempatan value ke dalam bin tergantung dari nilai arahnya, dan nilai yang diletakkan di bin adalah nilai magnitudenya.

Apabila besar nilai arah berada di antara nilai bin (misalnya nilai 16 yang berarti antara 0 dan 20), maka penempatan magnitude akan dibagi ke dalam kedua bin yang mengapit nilainya secara proporsional. Jika nilai arah lebih besar dari 160, maka perhitungan dilakukan antara 160 dan 180, dan magnitude dibagi ke bin 160 dan 0.

Jika perhitungan bin sudah selesai, maka nilai histogram tiap sel sudah didapatkan, nantinya akan disatukan menjadi histogram dari keseluruhan gambar.Langkah selanjutnya adalah melakukan normalisasi terhadap histogram yang sudah dibuat. Normalisasi dilakukan agar nilai histogram tidak berubah saat terkena perbedaan pencahayaan pada gambar. Jika tidak dilakukan, nilai histogram akan berubah karena beda pencahayaan dapaat mengubah nilai magnitude dari gradient.

Informasi histogram tersebut selanjutnya akan divisualisasi. Setelah itu, inputan yang sudah ditampung tersebut diinput untuk diproses algoritma Machine Learning untuk diklasifikasi. Berikut adalah contoh pseudocode dari HoG

```
for all pixel (i, j) in image x
  magnitude = abs(x(i-1, j-1)+x(i-1, j)+x(i-1, j+1)+x(i, j-1) -       x(i,j)*8+x(i,
j+1)+x(i+1, j-1)+x(i+1, j)+x(i+1, j+1))/8
  bin = x(i, j)/(256/9)
  sel = getSel(i, j)
  histogram (sel, bin) = histogram (sel, bin) + magnitude
 end
 for all sel y in image x
```

```
  deskriptor (y) = makeblock(histogram)
  deskriptor (y) = normalize(deskriptor(y))
end
for all deteksi z on window a in image x
  hasil = getHOG(deskriptor, a)
end
```

Potongan pseudocode tersebut menjelaskan cara kerja deskriptor *HoG* secara umum. Pertama untuk tiap piksel (i dan j) pada gambar x, definisikan magnitude, bin array, dan selnya. Lalu untuk tiap sel y pada gambar x, lakukan perhitungan histogramnya dan lakukan normalisasi dari hasil perhitungan tersebut. Terakhir ambil semua hasil deteksi z untuk perhitungan histogramnya menggunakan window a pada gambar x.

2.4. Support Vector Machine

Support Vector Machine (SVM) merupakan sebuah algoritma yang ditemukan pada 1963 oleh Vapnik dan Chervonenkis. Nantinya algoritma ini akan dikembangkan lagi oleh Boser, Guyon, dan Vapnik pada tahun 1992 (Chervonenkis, 2013). SVM adalah salah satu algoritma yang ada pada machine yang digunakan proses regresi dan/ atau klasifikasi data. Algoritma ini bekerja dengan cara mencari Hyperplane dari ruang N-dimensi yang membagi data kedalam suatu klasifikasi (Gandhi, 2018). Dengan kemampuan memberikan akurasi hasil yang signifikan dengan daya yang lebih kecil algoritma ini menjadi popular dalam dunia Machine Learning

Berdasarkan dataset yang digunakan untuk pembelajaran machine learning, ada dua jenis SVM yaitu linear dan non-linear. Dataset yang dapat dipisahkan secara linear dengan mudah disebut Linear SVM. Namun jika dataset yang digunakan cukup rumit dan mapping data dilakukan dengan cara mengganti dimensi, maka disebut Non-linear SVM. Gandhi (2018) dan Patel (2017) juga menyebutkan komponen penting pada SVM yaitu :

1) Linear Classifier

Fungsi – fungsi linear yang berisi nilai linear dari suatu klasifikasi. Digunakan untuk mencari *hyperplane*.

2) Hyperplane

Hyperplane merupakan batasan yang memisahkan kelompok data sesuai kelompok klasifikasinya. *Hyperplane* yang akan dipilih adaah yang memiliki jarak margin maksimal. Tergantung jumlah fitur, bentuk dari *Hyperplane* bisa berubah sesuai dimensinya

3) Support Vectors / Poin data

Poin – poin data yang posisinya paling dekat dengan *hyperplane* dan mempengaruhi posisi dan orientasinya.

***4)* Gradient**

Vector yang memiliki tiap bagian turunan dari suatu fungsi

***5)* Margin**

Merupakan jarak antar *support vectors / poin data*. Jarak nilai dari margin adalah [-1, 1] atau [0, 1]. Tergantung jaraknya, margin dibagi menjadi margin

kecil dan margin besar. Margin yang baik memiliki jarak yang cukup tanpa berada terlalu dekat dengan satu kelas.

Arliss (2020) mengatakan bahwa dalam algoritma *SVM,* kita akan menginginkan margin maksimal antara poin data dan hyperplane. Fungsi loss yang membantu memaksimalkan margin tersebut adalah hinge loss dengan rumus sebagai berikut

$$c\big(x, y, f(x)\big) = (1 - y * f(x)) \tag{2.3}$$

dimana :

y $\epsilon\{0, 1\}$

c = parameter regularisasi margin

cost adalah 0 jika value yang diprediksi dan value asli sama, jika tidak, maka akan dihitung loss valuenya

$$min_w \lambda \, ||w||^2 + \sum_{i=1}^{n} (1 - yi(xi, w)) \tag{6.4}$$

Setelah mendapatkan fungsi loss, bagian turunan akan diambil untuk mencari gradient

$$\frac{\delta}{\delta w_k} \lambda \, \big||w|\big|^2 = 2\lambda w_k \tag{6.5}$$

$$\frac{\delta}{\delta w_k} \lambda\big(1 - yi(xi, w)\big) = \begin{cases} 0, if\ yi(xi,w) \geq 1 \\ -yixik, else \end{cases} \tag{6.6}$$

selain itu, ketika tidak ada kesalahan dalam klasifikasi, selanjutnya hanya perlu memperbarui gradient dari c

$$w = w - a\,.\,(2\lambda w) \tag{6.7}$$

Namun jika terjadi misklasifikasi, selanjutnya tetap masukkan lossnya dengan parameter c saat memperbarui gradient

$$w = w + a\,.\,(yi\,.\,xi - 2\lambda w) \tag{6.8}$$

Berikut diberikan pseudocode untuk *SVM*

```
Define n features+1 as X and samplex+1 as Y
for each Y
    for each feature in Y
        read data
        normalize data
        store data into array[Y][X]
    end for
end for
for each feature
    count distance value
    for each sample
        count margin
        get hyperplane
    end for
    get class
end for
```

Penjelasan pseudocode adalah sebagai berikut. Untuk tiap fitur dan sampel, baca data yang ada pada fitur, lalu dinormalisasi agar mempermudah perhitungan secara setara sebelum dimasukkan ke array X dan Y. lalu untuk tiap fitur, hitung nilai jarak dari tiap sampel, dan untuk tiap sampel hitung besaran margin yang ditemukan, lalu dengan margin paling baik cari *hyperplane*nya. Terakhir kelas klasifikasi telah didapatkan.

Sementara, jika datasetnya ternyata non-linear, maka digunakan pseudocode seperti berikut. Langkah yang dilakukan kurang lebih sama seperti model linearnya, namun dalam pengerjaan *SVM non-linear* digunakan trik *kernel*. *Kernel* disini berfungsi untuk menambahkan dimensi baru pada vektor *SVM* agar sampel yang tadinya tidak bisa dihitung menggunakan *SVM linear* jadi bisa dihitung karena adanya tambahan dimensi.

```
Define n features+1 as X and samplex+1 as Y
for each Y
    for each feature in Y
        read data
        normalize data
        store data into array[Y][X]
    end for
end for
for each feature
    insert kernel
    count distance value
    for each sample
        count margin
        get hyperplane
    end for
    get class
end for
```