BAB II
LANDASAN TEORI

Bab ini menjelaskan mengenai teori-teori dasar yang digunakan pada penelitian.

2.1. Batik

Batik adalah proses pembuatan pola dengan teknik rintang celup menggunakan lilin panas yang diaplikasikan pada kain (Kudiya, Kreativitas dalam Desain Batik, 2019). Berdasarkan definisi tersebut, jika terdapat sebuah objek yang memiliki motif batik tetapi tidak dibuat menggunakan teknik rintang celup, objek tersebut tidak dapat disebut sebagai batik, hanya objek yang menggunakan motif batik. Motif batik secara bentuk dapat dibagi menjadi dua: geometris dan non-geometris (Kudiya, Kreativitas dalam Desain Batik, 2019). Motif geometris dibuat menggunakan pola yang berasal dari bentuk-bentuk geometri (garis, lingkaran, persegi, dll), sedangkan motif non-geometris dibuat menggunakan bentuk-bentuk objek sehari-hari (bunga, awan, hewan, dll).

Motif batik memiliki perbedaan motif dan warna dari setiap daerah, sehingga pada zamannya pengguna batik dapat diketahui asal daerahnya berdasarkan batik yang digunakan (Shidi & Suyoto, 2011). Salah satu batik daerah yang ada adalah batik Cierbon yang berasal dari Cirebon di provinsi Jawa Barat. Secara bahasa, Cirebon berasal dari dua kata sunda yaitu "ci" yang berarti air dan "rebon" yang merupakan sejenis udang berukuran kecil yang digunakan oleh masyarakat Cirebon untuk bahan produksi terasi (Kudiya, Djatmiko, Jusuf, & Atik, 2016). Hal ini dapat dilihat pada kenyataan bahwa Cirebon merupakan penghasil udang dan terasi. Salah satu variasi dari batik Cirebon adalah batik pesisiran.

Batik pesisiran Cirebon adalah batik yang tumbuh dan berkembang diluar dari daerah keraton. Batik ini dikembangkan oleh masyarakat jelata yang tidak berpatokan pada alam pikiran serta feodalisme aristokrasi Jawa yang menjadi dasar acuan batik keraton (Kudiya, Djatmiko, Jusuf, & Atik, 2016). Meskipun batik pesisiran tidak memiliki patokan dan aturan-aturan yang ketat seperti batik keraton, perkembangan batik pesisiran tetap dipengaruhi oleh corak-corak batik keraton karena batik tersebut tetap berkembang di daerah Cirebon yang memiliki keraton. Karena batik pesisiran tidak terikat dengan aturan-aturan keraton, maka batik

pesisiran memiliki tata warna dan corak yang lebih bebas sesuai keinginan masyarakat. Gambar 1 adalah contoh dari motif batik pesisiran Cirebon.

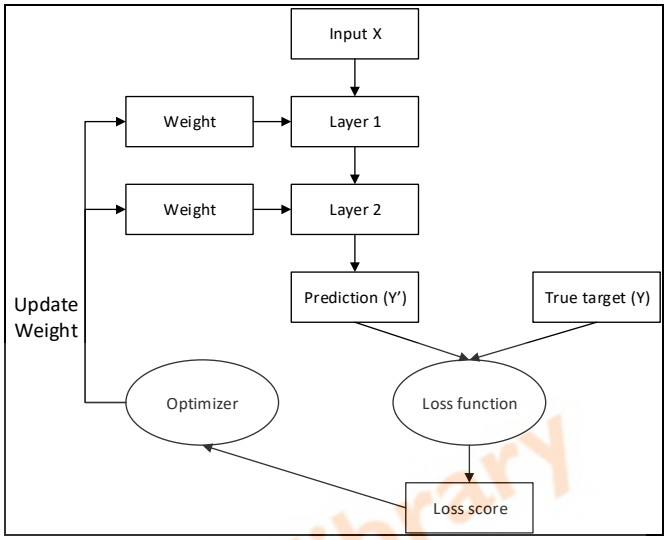
Gambar 1 Batik Pesisiran Cirebon Motif Megamendung (Kudiya, Djatmiko, Jusuf, & Atik, 2016)

2.2. *Deep Learning*

Deep learning merupakan proses representasi data dalam bentuk lapisan – lapisan informasi atau fitur dari data. Lapisan – lapis informasi ini bertambah kompleks dan mengandung informasi yang lebih banyak pada lapisan bawah disbanding lapisan atas (Chollet, 2018). Kedalaman informasi dari suatu model data direpresentasikan dalam jumlah lapisan yang ada, semakin banyak lapisan yang ada, semakin "dalam" model tersebut. Dengan adanya beberapa lapisan sebagai representasi data, sistem yang menggunakan deep learning dapat mengenali perbedaan data dalam proses klasifikasi.

Dalam melakukan representasi data berbentuk lapisan – lapisan informasi, biasanya digunakan model yang disebut *neural network*. Pada *neural network* informasi yang ada disusun sebagai lapisan yang bertumpuk (Chollet, 2018). Pada tumpukan ini, lapisan paling atas mengandung informasi paling sederhana dan lapisan paling bawah mengandung informasi paling rumit. Setiap lapisan mengandung beberapa informasi dengan kerumitan yang sama. Setiap informasi memiliki bobot berbeda yang digunakan untuk membedakan data masukan saat pengujian. Bobot ini dilatih terlebih dahulu agar saat digunakan, model dapat melakukan klasifikasi data.

Untuk melatih setiap lapisan informasi pada model neural network digunakan sebuah algoritma bernama *backpropagation*. Secara umum, algoritma *backpropagation* bekerja dengan cara memperbaiki bobot dari setiap lapisan dengan cara menghitung tingkat kesalahan yang terjadi dari setiap proses klasifikasi data. Proses tersebut dapat dilihat pada Gambar 2.



Gambar 2 Proses *Backpropagation* (Chollet, 2018)

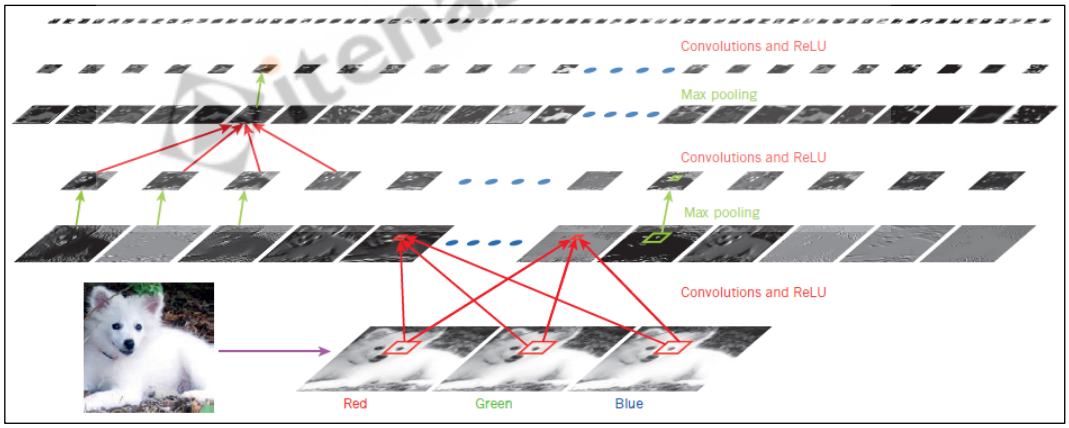
Proses *backpropagation* dilakukan dengan cara menghitung perbedaan antara hasil prediksi dan nilai data sesungguhnya lalu menggunakan nilai perbedaan itu untuk melakukan optimalisasi bobot (Chollet, 2018). Pertama, dilakukan prediksi menggunakan bobot acak yang menghasilkan nilai Y'. Setelah itu, nilai prediksi dibandingkan dengan nilai data sesungguhnya (Y) dan dihitung *loss score* menggunakan *loss function*. Nilai *loss score* digunakan untuk memperbaiki bobot oleh *optimizer* agar prediksi selanjutnya lebih akurat. Proses ini dilakukan berkali-kali hinga nilai bobot sudah stabil.

2.3. *Convolutional Neural Network*

Convolutional neural network (CNN) adalah salah satu variasi dari *neural network* yang dirancang untuk memproses data berbentuk sejumlah *array* (LeCun, Bengio, & Hinton, 2015). Sebagai contoh, data berupa citra merupakan data berbentuk tiga buah *array* dengan masing masing *array* mewakili warna merah, hijau, dan biru (untuk citra dengan format RGB). *Array* tersebut diolah menggunakan proses konvolusi pada lapisan konvolusi di dalam *convolutional*

neural network (biasa juga disebut *hidden layer*). Proses konvolusi yang terjadi pada *hidden layer* berfungsi untuk mendapatkan fitur dari citra masukan (Chollet, 2018). Setiap *hidden layer* yang ada memiliki beberapa *kernel*/*filter* yang berfungsi untuk melakukan deteksi fitur. Jumlah *kernel* yang ada tergantung dari arsitektur CNN yang digunakan. Arsitektur CNN yang tersedia antara lain adalah AlexNet, VGG, GoogLeNet, dan ResNet.

Asitektur CNN secara umum mengandung *convolutional layer*, *non-linearity layer*, *pooling layer*, *fully connected layer*, dan *output layer* (LeCun, Bengio, & Hinton, 2015). Pada *convolutional layer*, dilakukan konvolusi data menggunakan filter yang ada pada lapisan tersebut yang berfungsi untuk mendeteksi fitur dan memperkecil dimensi data. Setelah itu hasil konvolusi dihitung menggunakan fungsi non-lienarity untuk mengambil bobotnya. Salah satu fungsi yang dapat digunakan adalah fungsi ReLU (rectified linear unit). Hasil data dari proses tersebut adalah *feature map*. Data tersebut selanjutnya masuk ke *pooling layer* dan menuju *convolution layer* selanjutnya. Proses ini ditumpuk beberapa kali hingga akhirnya mencapai *output layer*. Arsitektur CNN diilustrasikan pada Gambar 3.



Gambar 3 Arsitektur CNN (LeCun, Bengio, & Hinton, 2015)

Pada Gambar 3 terdapat sebuah citra yang dimasukkan kedalam CNN. Citra tersebut memiliki 3 kanal yaitu kanal *red*, *green*, dan *blue*. Masing masing kanal tersebut melewati tahap konvolusi dan aktivasi (menggunakan fungsi ReLU) untuk mendapatkan *feature map* dari lapisan pertama. Setelah itu, dilakukan *max-pooling* dan konvolusi ke lapisan selanjutnya hingga lapisan terakhir pada CNN tersebut.

CNN memiliki penggunaan yang luas pada *deep learning* (Nielsen, 2015), salah satunya adalah untuk melakukan *deep style transfer* (Gatys, Ecker, & Bethge, 2015). CNN dapat digunakan pada *deep style transfer* karena objek pada *deep style transfer* merupakan objek dua dimensi yaitu citra.

2.3.1. VGG19

VGG adalah sebuah arsitektur CNN yang dirancang oleh (Simonyan & Zisserman, 2015). Konfigurasi arsitektur VGG yang digunakan pada penelitian ini adalah VGG19 (konfigurasi E pada gambar 4 yang memiliki jumlah lapisan sebanyak 19 buah (16 lapisan konvolusi dan 3 lapisan *fully connected*). VGG19 dibentuk menggunakan tumpukan lapisan konvolusi dengan *filter* berukuran 3x3. Setiap lapisan konvolusi dilengkapi dengan fungsi *non-linearity* berupa ReLU (*Rectified Linear unit*) (Krizhevsky, Sutskever, & Hinton, 2012). Proses *max pooling* dilakukan dengan dimensi *filter* 2x2 dengan jumlah *stride* 2.

Pada VGG19 terdapat 5 blok lapisan konvolusi yang dipisahkan oleh lapisan *max-pooling*. Blok 1 melakukan konvolusi sebanyak 2 kali, masing-masing konvolusi diberi nama "block1_conv1" dan "block1_conv2". Penamaan ini digunakan untuk blok 2 hingga blok 5 dengan format penamaan yang sama.

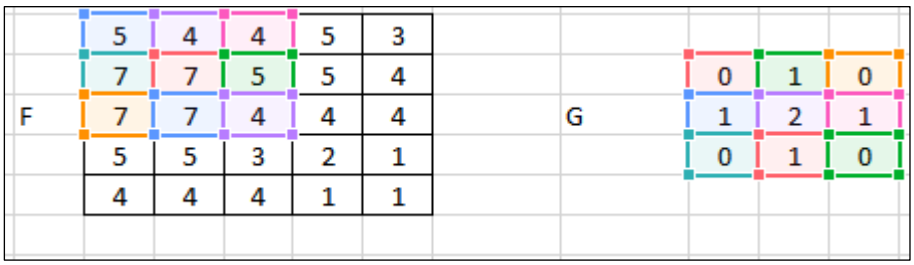
Tiga lapisan *fully connected* (FC) terdapat setelah tumpukan lapisan konvolusi dan *max pooling*. Dua lapisan *fully connected* memliki 4096 parameter. Lapisan tersebut diikuti oleh lapisan *fully connected* dengan jumlah parameter sebanyak 1000. Lapisan terakhir dari arsitektur ini adalah *soft-max*.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Gambar 4 Arsitektur VGG (Simonyan & Zisserman, 2015)

2.3.2. Konvolusi

Proses konvolusi merupakan proses penerapan filter yang terdapat di dalam setiap lapisan konvolusi terhadap citra untuk mencari fitur yang ada pada citra tersebut (LeCun, Bengio, & Hinton, 2015). Hasil dari proses konvolusi dapat disebut dengan *feature map* (LeCun, Bengio, & Hinton, 2015) karena setiap hasil konvolusi merupakan representasi fitur yang dideteksi oleh filter yang bersangkutan. Proses perhitungan konvolusi diilustrasikan pada Gambar 5.



Gambar 5 Ilustrasi Operasi Konvolusi 1

Konvolusi dilakukan menggunakan matriks G sebagai filter dan matriks F sebagai citra yang dikonvolusi. Untuk melakukan konvolusi, Letakkan filter pada pojok kiri atas citra. Setelah itu kalikan nilai filter dengan nilai citra dan jumlahkan hasil perkalian tersebut. Sebagai contoh, untuk mendapatkan hasil konvolusi, lakukan operasi:

- (5*0) + (4*1) + (4*0) + (7*1) + (7*2) + (5*1) + (7*0) + (7*1) + (7*0) = 37

Hasil operasi tersebut diletakkan pada pojok kiri atas citra hasil konvolusi. Setelah itu, geser posisi filter ke kiri sebanyak 1 kolom dan lakukan proses tersebut kembali. Jika posisi filter sudah mencapai pojok kanan, kembalikan posisi filter ke pojok kiri dan turunkan posisi sebanyak 1 baris. Lakukan proses ini hingga seluruh nilai pada citra telah dioperasikan menggunakan filter. Hasil operasi konvolusi dapat dilihat pada Gambar 6.

	37	30	28
X	37	27	23
	29	21	13

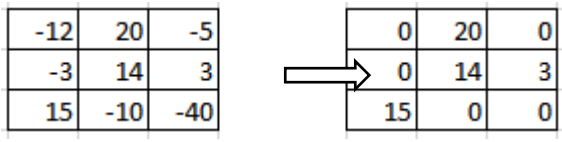
Gambar 6 Ilustrasi Operasi Konvolusi 2

2.3.3. ReLU

ReLU (*Rectified Linear Unit*) merupakan fungsi aktivasi *non-linearity* yang berfungsi untuk mendapatkan nilai *output* dari setiap lapisan konvolusi (Krizhevsky, Sutskever, & Hinton, 2012). Proses ini dapat dilakukan dengan persamaan (1)

$$f(x) = \max(0, x) \quad (1)$$

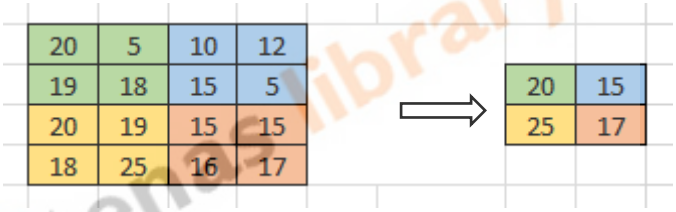
Dimana f(x) adalah nilai dari ReLU dan x adalah nilai elemen *matriks* citra. Rumus tersebut berfungsi untuk mengubah nilai elemen x menjadi 0 jika x<0 dan tidak mengubah nilai x jika x>0. Dengan cara ini nilai negatif yang berasal dari proses konvolusi dapat dihilangkan dari citra. Proses penggunaan ReLU diilustrasikan pada Gambar 7.



Gambar 7 Ilustrasi ReLU

2.3.4. *Max-Pooling*

Max-pooling merupakan proses yang berfungsi untuk mengecilkan ukuran citra dengan cara mengambil nilai paling besar dari *pixel* yang berdekatan. Metode ini digunakan karena pixel yang berdekatan merepresentasikan fitur yang sama (LeCun, Bengio, & Hinton, 2015). Karena alasan tersebut, ukuran citra dapat dikecilkan dengan cara mengambil satu pixel paling besar dari filter yang diterapkan pada citra. Ukuran filter yang digunakan berukuran z x z dengan *stride* biasanya bernilai s = z (Krizhevsky, Sutskever, & Hinton, 2012). Proses *max-pooling* diilustraskan pada Gambar 8.



Gambar 8 Ilustrasi proses *max-pooling*

Pada Gambar 8 diilustrasikan proses *max-pooling* dengan ukuran *filter* 2x2 dan *stride* = 2, sesuai dengan ukuran yang digunakan pada VGG19 (Simonyan & Zisserman, 2015). Pertama, terapkan filter *max-pooling* pada koordinat (0,0) pada citra. Ambil nilai terbesar dari [20, 5, 19, 18]. Nilai tersebut menjadi nilai pada koordinat (0,0) di citra hasil. Lalu geser filter sebanyak nilai *stride*. Ambil nilai terbesar dari [10, 12, 15, 5] lalu simpan nilai tersebut pada koordinat (0,1) di citra hasil. Lakukan proses ini hingga semua *pixel* telah diambil nilai *max-pool*. Hasil proses *max-pooling* dari citra 4x4 dengan *filter* berukuran 2x2 dan *stride* = 2 adalah citra berukuran 2x2.

2.4. *Deep Style Transfer*

Deep style transfer (Gatys, Ecker, & Bethge, 2015) merupakan salah satu variasi dari *generative deep learning* (Chollet, 2018). Selain *deep style transfer*, terdapat juga *deep dream* yang diajukan olah (Mordvintsev, Olah, & Tyka, 2015). Pada

metode *deep dream*, *neural network* diberikan fitur atau tekstur dari beberapa citra. Setelah itu, citra konten yang dimasukkan diubah teksturnya sesuai dengan tekstur yang ada pada *neural network* yang telah dilatih. Metode ini telah diteliti oleh (McCaig, DiPaola, & Gabora, 2016), (DiPaola, Gabora, & McCaig, 2018), dan (Utz & DiPaola, 2019). Penelitian oleh (McCaig, DiPaola, & Gabora, 2016) dan (DiPaola, Gabora, & McCaig, 2018) dilakukan menggunakan citra serta *neural network* yang telah dilatih untuk meneliti nilai seni dari hasil *deep dream*. Pada penelitian yang dilakukan oleh (Utz & DiPaola, 2019), hasil *deep dream* dijadikan sebuah video dan diuji nilai seninya kepada beberapa partisipan.

Metode *Deep style transfer* (Gatys, Ecker, & Bethge, 2015), berbeda dengan *deep dream*, tidak melatih seluruh *neural netwok* untuk dijadikan tekstur, melainkan mengambil tekstur dari sebuah citra lalu menggabungkannya dengan konten dari citra lainnya. Hal ini dilakukan dengan memanfaatkan *convolutional neural network* (CNN) untuk menghasilkan sebuah citra dari citra konten dan citra tekstur. Citra hasil merupakan citra konten yang diperbarui berdasarkan nilai *loss* yang didapat antara citra hasil dan citra konten serta antara citra hasil dan citra tekstur. Berdasarkan penelitian yang dilakukan oleh (Gatys, Ecker, & Bethge, 2015), dalam proses *Deep style transfer* perlu dilakukan perhitungan *content loss*, *style loss*, dan total *loss*. Penelitian ini menggunakan VGG19 (Simonyan & Zisserman, 2015) sebagai arsitektur CNN dan Adam (Kingma & Ba, 2017) sebagai *optimizer*.

2.4.1. *Resize*

Proses *resize* dilakukan untuk mengubah ukuran citra yang diproses oleh *deep style transfer*. Pada saat melakukan *resize*, perbandingan antara lebar dengan tinggi citra harus dipertahankan untuk menjaga seluruh fitur yang ada pada citra. Penelitian ini memberikan tinggi baru untuk citra masukannya dan menghitung lebar baru supaya menyesuaikan perbandingan tinggi dan lebar citra. Menurut (Chollet, 2018), penghitungan nilai lebar baru dapat dihitung dengan persamaan (2):

$$new\ width = \frac{image\ width\ x\ new\_height}{image\_height} \quad (2)$$

2.4.2. *Content Loss*

Content loss merupakan perhitungan yang dilakukan untuk menghitung perbedaan konten antara citra konten dan citra hasil. Proses ini dilakukan dengan menghitung perbedaan nilai *feature map* dari citra konten dan citra hasil. Menurut (Gatys, Ecker, & Bethge, 2015) nilai perbedaan dari dua *feature map* dapat dihitung dengan persamaan (3)

$$L_{(\vec{g},\vec{c},l)}=\frac{1}{2}\sum_{ij}(G_{ij}-c_{ij})^2 \quad (3)$$

Dimana L adalah *content loss*, $\vec{g}$ dan $\vec{c}$ berturut – turut adalah citra hasil dan citra konten, G dan C adalah *feature map* dari citra hasil dan citra konten pada lapisan ke-l.

2.4.3. *Style Loss*

Style loss adalah perhitungan yang digunakan untuk menghitung perbedaan "tekstur" antara citra tekstur dan citra hasil. Berdasarkan penelitian yang dilakukan oleh (Gatys, Ecker, & Bethge, 2015), perbedaan ini dapat dihitung dengan membuat matriks *gram* dari *feature map* yang ada pada setiap lapisan CNN. Matriks *gram* dapat dibuat dengan cara menghitung *inner product* dari *feature map* yang telah dijadikan bentuk vektor. Nilai matriks *gram* dapat dihitung dengan persamaan (4)

$$G_{ij}^{l} = \sum_k F_{ik}F_{jl} \quad (4)$$

Setelah mendapat matriks *gram*, nilai *style loss* per-lapisan dapat dihitung dengan persamaan (5)

$$\mathrm{E}_l = \frac{1}{4N_l^2M_l^2}\sum_{ij}(G_{ij}^{l} - S_{ij}^{l})^2 \quad (5)$$

Dengan E adalah *style loss*, N adalah jumlah filter pada lapisan ke-l, M adalah perkalian dari tinggi dan lebar filter, G dan S adalah matriks *gram* dari citra hasil dan citra tekstur.

2.4.4. Total *Loss*

Total *loss* dihitung dengan cara menghitung terlebih dahulu faktor kontribusi dari *loss* dan *style loss*. Total kontribusi dapat dihitung dengan persamaan (6)

$$Total\ Loss\ Contribution = \sum_l (w\ x\ Loss) \quad (6)$$

Dengan l adalah lapisan ke-l dan w adalah factor kontribusi bobot *loss*. Nilai w dimasukkan ke dalam sistem sebagai nilai statis. Setelah nilai kontribusi didapatkan, total *loss* dapat dihitung dengan cara menjumlahkan kontribusi *content loss* dan kontribusi *style loss*.