UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Team project ©2017
Dony Pratidana S. Hum | Bima Agus Setyawan S. IIP

BAB III

METODE PENELITIAN

3.1 Studi Pustaka

Pada tahap ini, penelitian dilakukan dengan mencari, mengumpulkan, mempelajari referensi-referensi terkait yang mendukung, dan mempunyai hubungan dengan penelitian ini. Referensi dapat berupa *paper*, jurnal ilmiah artikel, buku, dan *source code* yang berkaitan dengan penelitian ini, seperti konsep *Generative Adversarial Network* (*GAN*) dan *source code* mengenai implementasi *GAN*. Sumber yang dijadikan untuk referensi diantaranya berasal dari arXiv.org untuk sumber *paper* dan Github.com untuk *source code*.

3.2 Spesifikasi Perangkat Keras

Pada penelitian ini dibutuhkan perangkat keras yang mendukung *framework* Tensorflow-gpu. *Training GAN* dilakukan menggunakan instrumen perangkat keras sebagai berikut:

- Prosesor Intel Pentium G3258
- Kartu grafis ASUS Nvidia GTX 1050 Ti
- RAM 8 GB *Dual Channel*

3.3 Spesifikasi Perangkat Lunak

Selain perangkat keras dibutuhkan juga perangkat lunak yang dapat mendukung kerangka kerja *Tensorflow* sehingga dapat menggunakan kemampuan komputasi dengan kartu grafis Nvidia. Perangkat lunak yang digunakan dalam penelitian ini sebagai berikut:

Rancang bangun paket..., Richard Dharmawan, FTI UMN, 2019

- Sistem Operasi Windows 10 *build* 17134
- Python Anaconda 3.7.1
- Nvidia driver versi 419.17
- CUDA 9.0
- cuDNN 7 untuk CUDA 9.0
- Tensorflow GPU versi 1.5.0

3.4 Persiapan *Environment*

Persiapan *environtment* dilakukan untuk memasang perangkat lunak sesuai dengan spesifikasi. Pertama diawali dengan memasang *environtment compute library* Nvidia yang terdiri dari *driver* Nvidia, *library* CUDA dan cuDNN. Selanjutnya, memasang Python versi Anaconda, lalu memasang *virtual environtment* Python yang didalamnya akan dipasang Python versi 3.6 dan *library* pendukung Tensorflow seperti Numpy dan Scipy untuk komputasi, Utils untuk membaca dan menulis *file*, dan Pillow untuk pemrosesan gambar. Terakhir setelah semua *dependency library* sudah dipasang dilanjut dengan memasang Tensorflow-GPU.

3.5 Implementasi dan Training GAN

Implementasi pertama yaitu penulisan kode menggunakan bahasa Python. Pada implementasi ini ada tiga fungsi utama yaitu *generator*, *discriminator*, dan *train*. Kode yang dituliskan dalam penelitian ini mengacu pada arsitektur DCGAN [6]. *Generator* merupakan sebuah fungsi yang menerima masukan berupa *input vector* yang bernilai acak dengan dimensi sebesar 100, dan mengeluarkan hasil berupa gambar yang nantinya akan dinilai apakah menghasilkan motif batik atau tidak, didalam fungsi ini terdapat *deconvolutional layer* sebanyak lima *layer*. *Layer* pertama akan menerima *input vector*, kemudian akan diproyeksikan menjadi

Rancang bangun paket..., Richard Dharmawan, FTI UMN, 2019

vektor dengan ukuran 8x8x256. Hasilnya akan diproyeksikan oleh *layer* kedua menjadi ukuran 16x16x128. Hasil *layer* kedua diproyeksikan kembali menjadi 32x32x64 oleh *layer* ketiga, kemudian menjadi 64x64x32 oleh *layer* keempat, dan terakhir menjadi 128x128x32. Tiap *layer* memiliki fungsi aktivasi yaitu ReLU kecuali pada layer terakhir menggunakan Tan Hiperbolik. Hasil kodenya ada pada gambar 3.1.

```python
def generator(input, random_dim, is_train, reuse=False):
    c4, c8, c16, c32, c64 = 512, 256, 128, 64, 32 # channel num
    s4 = 4
    output_dim = CHANNEL  # RGB image
    with tf.variable_scope('gen') as scope:
        if reuse:
            scope.reuse_variables()
        w1 = tf.get_variable('w1', shape=[random_dim, s4 * s4 * c4], dtype=tf.float32,
                             initializer=tf.truncated_normal_initializer(stddev=0.02))
        b1 = tf.get_variable('b1', shape=[c4 * s4 * s4], dtype=tf.float32,
                             initializer=tf.constant_initializer(0.0))
        flat_conv1 = tf.add(tf.matmul(input, w1), b1, name='flat_conv1')
        #Convolution, bias, activation, repeat!
        conv1 = tf.reshape(flat_conv1, shape=[-1, s4, s4, c4], name='conv1')
        bn1 = tf.contrib.layers.batch_norm(conv1, is_training=is_train, epsilon=1e-5, decay = 0.9,  updates_collections=None, scope='bn1')
        act1 = tf.nn.relu(bn1, name='act1')
        # 8*8*256
        #Convolution, bias, activation, repeat!
        conv2 = tf.layers.conv2d_transpose(act1, c8, kernel_size=[5, 5], strides=[2, 2], padding="SAME",
                                           kernel_initializer=tf.truncated_normal_initializer(stddev=0.02),
                                           name='conv2')
        bn2 = tf.contrib.layers.batch_norm(conv2, is_training=is_train, epsilon=1e-5, decay = 0.9,  updates_collections=None, scope='bn2')
        act2 = tf.nn.relu(bn2, name='act2')
        # 16*16*128
        conv3 = tf.layers.conv2d_transpose(act2, c16, kernel_size=[5, 5], strides=[2, 2], padding="SAME",
                                           kernel_initializer=tf.truncated_normal_initializer(stddev=0.02),
                                           name='conv3')
        bn3 = tf.contrib.layers.batch_norm(conv3, is_training=is_train, epsilon=1e-5, decay = 0.9,  updates_collections=None, scope='bn3')
        act3 = tf.nn.relu(bn3, name='act3')
        # 32*32*64
        conv4 = tf.layers.conv2d_transpose(act3, c32, kernel_size=[5, 5], strides=[2, 2], padding="SAME",
                                           kernel_initializer=tf.truncated_normal_initializer(stddev=0.02),
                                           name='conv4')
        bn4 = tf.contrib.layers.batch_norm(conv4, is_training=is_train, epsilon=1e-5, decay = 0.9,  updates_collections=None, scope='bn4')
        act4 = tf.nn.relu(bn4, name='act4')
        # 64*64*32
        conv5 = tf.layers.conv2d_transpose(act4, c64, kernel_size=[5, 5], strides=[2, 2], padding="SAME",
                                           kernel_initializer=tf.truncated_normal_initializer(stddev=0.02),
                                           name='conv5')
        bn5 = tf.contrib.layers.batch_norm(conv5, is_training=is_train, epsilon=1e-5, decay = 0.9,  updates_collections=None, scope='bn5')
        act5 = tf.nn.relu(bn5, name='act5')

        #128*128*3
        conv6 = tf.layers.conv2d_transpose(act5, output_dim, kernel_size=[5, 5], strides=[2, 2], padding="SAME",
                                           kernel_initializer=tf.truncated_normal_initializer(stddev=0.02),
                                           name='conv6')
        # bn6 = tf.contrib.layers.batch_norm(conv6, is_training=is_train, epsilon=1e-5, decay = 0.9,  updates_collections=None, scope='bn6')
        act6 = tf.nn.tanh(conv6, name='act6')
        return act6
```

Gambar 3.1: Potongan kode *Generator*

Discriminator merupakan fungsi kebalikan dari *Generator* didalamnya terdapat empat *convolutional layer*. *Input* dari *discriminator* berupa hasil keluaran *Generator*. Setiap *layer* akan mengubah dimensi input menjadi 32x32x64, 16x16x128, 8x8x256, dan 4x4x512. Fungsi aktivasi yang digunakan yaitu LeakyReLU untuk semua *layer* kecuali *layer output* menggunakan fungsi Sigmoid. Hasil kodenya ada pada gambar 3.2.

Rancang bangun paket..., Richard Dharmawan, FTI UMN, 2019

UNIVERSITAS
MULTIMEDIA
NUSANTARA

```python
def discriminator(input, is_train, reuse=False):
    c2, c4, c8, c16 = 64, 128, 256, 512  # channel num: 64, 128, 256, 512
    with tf.variable_scope('dis') as scope:
        if reuse:
            scope.reuse_variables()

        #Convolution, activation, bias, repeat!
        conv1 = tf.layers.conv2d(input, c2, kernel_size=[5, 5], strides=[2, 2], padding="SAME",
                                 kernel_initializer=tf.truncated_normal_initializer(stddev=0.02),
                                 name='conv1')
        bn1 = tf.contrib.layers.batch_norm(conv1, is_training = is_train, epsilon=1e-5, decay = 0.9,  updates_collections=None, scope = 'bn1')
        act1 = lrelu(conv1, n='act1')
         #Convolution, activation, bias, repeat!
        conv2 = tf.layers.conv2d(act1, c4, kernel_size=[5, 5], strides=[2, 2], padding="SAME",
                                 kernel_initializer=tf.truncated_normal_initializer(stddev=0.02),
                                 name='conv2')
        bn2 = tf.contrib.layers.batch_norm(conv2, is_training=is_train, epsilon=1e-5, decay = 0.9,  updates_collections=None, scope='bn2')
        act2 = lrelu(bn2, n='act2')
        #Convolution, activation, bias, repeat!
        conv3 = tf.layers.conv2d(act2, c8, kernel_size=[5, 5], strides=[2, 2], padding="SAME",
                                 kernel_initializer=tf.truncated_normal_initializer(stddev=0.02),
                                 name='conv3')
        bn3 = tf.contrib.layers.batch_norm(conv3, is_training=is_train, epsilon=1e-5, decay = 0.9,  updates_collections=None, scope='bn3')
        act3 = lrelu(bn3, n='act3')
         #Convolution, activation, bias, repeat!
        conv4 = tf.layers.conv2d(act3, c16, kernel_size=[5, 5], strides=[2, 2], padding="SAME",
                                 kernel_initializer=tf.truncated_normal_initializer(stddev=0.02),
                                 name='conv4')
        bn4 = tf.contrib.layers.batch_norm(conv4, is_training=is_train, epsilon=1e-5, decay = 0.9,  updates_collections=None, scope='bn4')
        act4 = lrelu(bn4, n='act4')

        # start from act4
        dim = int(np.prod(act4.get_shape()[1:]))
        fc1 = tf.reshape(act4, shape=[-1, dim], name='fc1')


        w2 = tf.get_variable('w2', shape=[fc1.shape[-1], 1], dtype=tf.float32,
                             initializer=tf.truncated_normal_initializer(stddev=0.02))
        b2 = tf.get_variable('b2', shape=[1], dtype=tf.float32,
                             initializer=tf.constant_initializer(0.0))

        # wgan just get rid of the sigmoid
        logits = tf.add(tf.matmul(fc1, w2), b2, name='logits')
        # dcgan
        acted_out = tf.nn.sigmoid(logits)
        return logits #, acted_out
```

Gambar 3.2: Potongan kode *discriminator*

Fungsi *train* merupakan fungsi utama yang akan dijalankan ketika program dieksekusi. Fungsi ini akan menjalankan *training* sesuai dengan EPOCH yang ditentukan dan menulis hasil keluaran dalam format .jpg. Bagian utama fungsi ini terlihat pada gambar 3.3.

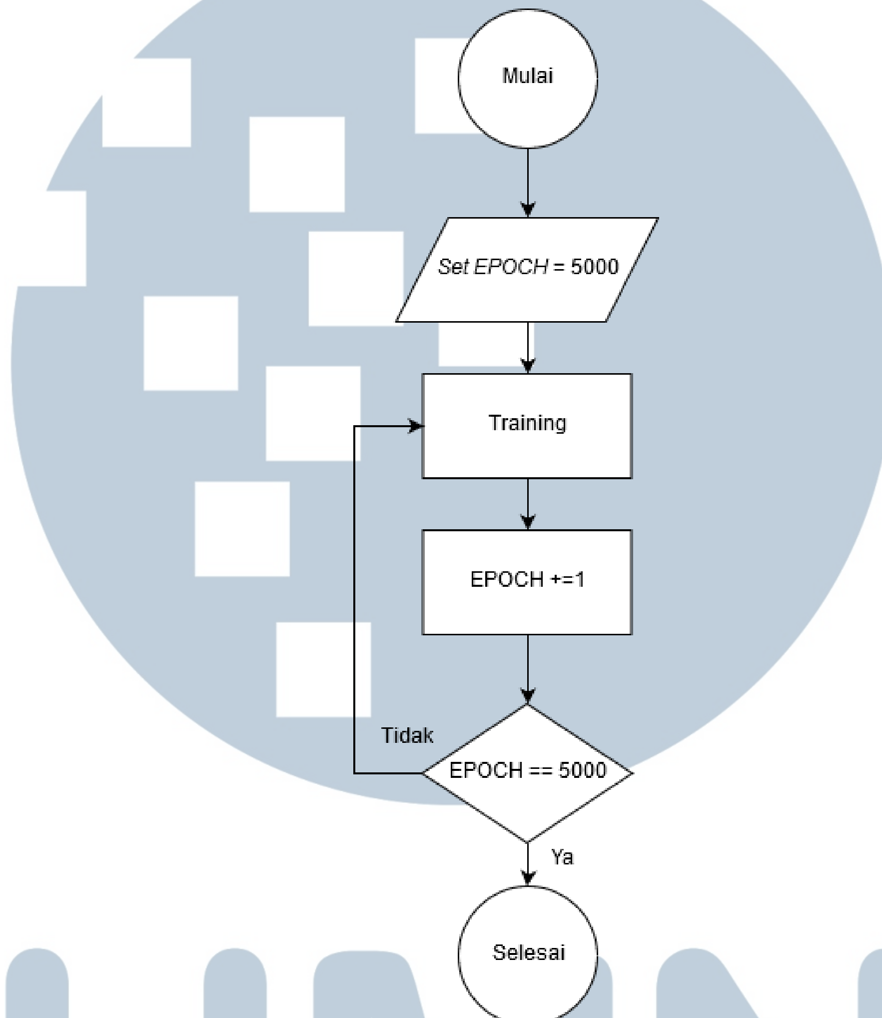
UNIVERSITAS MULTIMEDIA NUSANTARA

```
print('total training sample num:%d' % samples_num)
print('batch size: %d, batch num per epoch: %d, epoch num: %d' % (batch_size, batch_num, EPOCH))
print('start training...')
for i in range(EPOCH):
    print("Running epoch {}/{}...".format(i, EPOCH))
    for j in range(batch_num):
        print(j)
        d_iters = 5
        g_iters = 1

        train_noise = np.random.uniform(-1.0, 1.0, size=[batch_size, random_dim]).astype(np.float32)
        for k in range(d_iters):
            print(k)
            train_image = sess.run(image_batch)
            #wgan clip weights
            sess.run(d_clip)

            # Update the discriminator
            _, dLoss = sess.run([trainer_d, d_loss],
                                feed_dict={random_input: train_noise, real_image: train_image, is_train: True})

        # Update the generator
        for k in range(g_iters):
            # train_noise = np.random.uniform(-1.0, 1.0, size=[batch_size, random_dim]).astype(np.float32)
            _, gLoss = sess.run([trainer_g, g_loss],
                                feed_dict={random_input: train_noise, is_train: True})

        # print 'train:[%d/%d],d_loss:%f,g_loss:%f' % (i, j, dLoss, gLoss)

    # save check point every 500 epoch
    if i%500 == 0:
        if not os.path.exists('./model/' + version):
            os.makedirs('./model/' + version)
        saver.save(sess, './model/' +version + '/' + str(i))
    if i%50 == 0:
        # save images
        if not os.path.exists(newPoke_path):
            os.makedirs(newPoke_path)
        sample_noise = np.random.uniform(-1.0, 1.0, size=[batch_size, random_dim]).astype(np.float32)
        imgtest = sess.run(fake_image, feed_dict={random_input: sample_noise, is_train: False})
        # imgtest = imgtest * 255.0
        # imgtest.astype(np.uint8)
        save_images(imgtest, [8,8] ,newPoke_path + '/epoch' + str(i) + '.jpg')

        print('train:[%d],d_loss:%f,g_loss:%f' % (i, dLoss, gLoss))
coord.request_stop()
coord.join(threads)
```

Gambar 3.3: Bagian utama fungsi *train*

Langkah selanjutnya yaitu *training*. *Training* dilakukan menggunakan data motif batik yang didapat dari penelitian yang dilakukan oleh Yohanes Gultom[2]. Pada penelitian tersebut motif batik yang digunakan yaitu Kawung, Ceplok, Nitik, dan Parang. Data motif batik diambil dari sumber tersebut karena untuk mempersingkat waktu penelitian sehingga tidak perlu mencari gambar motif yang cukup banyak, motif yang didapat lebih dari satu, dan sudah terkelompokan berdasarkan jenisnya. Hasil dari *training* yang berupa gambar akan dihitung berapa banyak motif yang memenuhi kriteria yang sudah ditentukan. Dari gambar keluaran akan terlihat apakah *network* masuk ke dalam *"mode collapse"*.

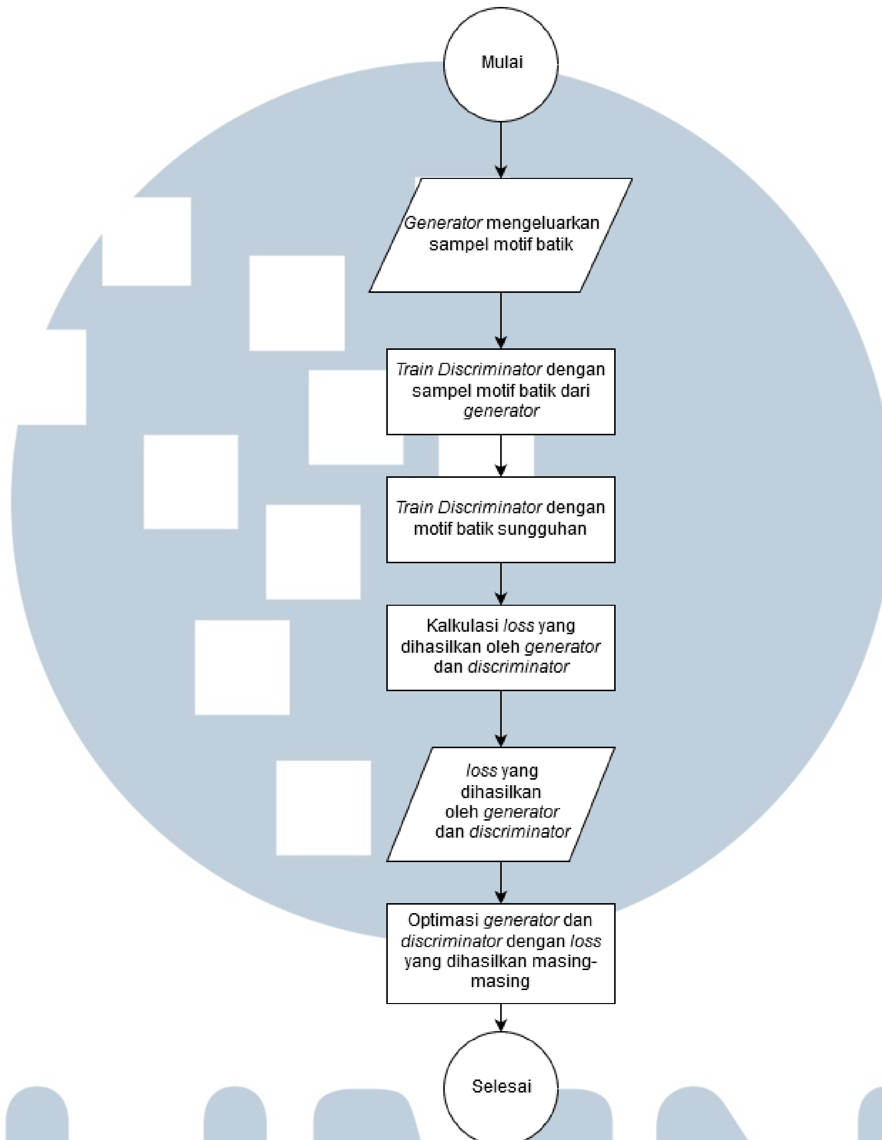
Training akan dilakukan sebanyak 5000 *EPOCH* atau 5000 iterasi. Diagram alur dari *training* dapat dilihat pada gambar di bawah.



Gambar 3.4: *flowchart training*

Input dalam proses *training* berupa gambar sampel motif batik sungguhan dengan keluaran berupa gambar yang dihasilkan oleh *generator*. Diagram alur dari proses *training* dapat dilihat pada gambar di bawah.

UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.5: *flowchart* proses *training*

3.6 Dokumentasi

Dokumentasi dilakukan dengan mecatat tahapan-tahapan yang sudah dilakukan selama proses perancangan dan memberikan keterangan pada *source code* berupa penjelasan fungsi dan variabel yang digunakan dalam bentuk komentar. Dokumentasi diperlukan untuk memudahkan peneliti lain mempelajari dan atau mengimplemntasikan penelitian ini kedalam aplikasi lain serta mendukung penelitian lebih lanjut terkait dengan penelitian ini.

Rancang bangun paket..., Richard Dharmawan, FTI UMN, 2019