Introduction - IMAGE TO SKETCH CONVERSION

Autoencoder is special type of deep learning architecture consisting of two networks a) encoder b)decoder. Encoder can be fully connected dense neural network or Convolution neural network. Encoder is used to downsample our original sample image into latent vector by passing image through Convolution layers and maxpool layer.Similary,decoder also can be fully connected neural network or Convolution neural network, decoder is used to upsample the latent vector downsampled by encoder. This upsampled latent vector is compared with the original input and reconstruction loss is calculated. Backpropagation is used to minimized this reconstruction loss. Simple autoencoder can be used for Domain transformation, denoising images, image colorization, anamoly detection etc. Here we are going to train my autoencoder model to generate sketch of the input image.

Objective:

To convert image to sketch using autoencoder

Import Necessary Libraries

```
import numpy as np
import tensorflow as tf
import keras
from keras.layers import Dense, Conv2D, MaxPool2D, UpSampling2D, 
Dropout, Input
from tensorflow.keras.utils import img_to_array
import matplotlib.pyplot as plt
import cv2
from tqdm import tqdm
import os
import re
```

Load data

This dataset consist of 188 image and their corresponding sketches. As these images aren't enough for training our autoencoder model, we have augmented them using open cv library. After Augmentation we have got around 1500 images, these 1500 images. These images are converted into array and are stored in the list.

```
# to get the files in proper order
def sorted_alphanumeric(data):
    convert = lambda text: int(text) if text.isdigit() else 
text.lower()
    alphanum_key = lambda key: [convert(c) for c in re.split('([0-
9]+)',key)]
    return sorted(data,key = alphanum_key)


# defining the size of image
```

```python
SIZE = 256

image_path = '/kaggle/input/cuhk-face-sketch-database-cufs/cuhk-face-sketch-database-cufs/photos'
img_array = []

sketch_path = '/kaggle/input/cuhk-face-sketch-database-cufs/cuhk-face-sketch-database-cufs/sketches'
sketch_array = []

image_file = sorted_alphanumeric(os.listdir(image_path))
sketch_file = sorted_alphanumeric(os.listdir(sketch_path))


for i in tqdm(image_file):
    image = cv2.imread(image_path + '/' + i,1)

    # as opencv load image in bgr format converting it to rgb
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # resizing images
    image = cv2.resize(image, (SIZE, SIZE))

    # normalizing image
    image = image.astype('float32') / 255.0

    #appending normal normal image
    img_array.append(img_to_array(image))
    # Image Augmentation

    # horizontal flip
    img1 = cv2.flip(image,1)
    img_array.append(img_to_array(img1))
     #vertical flip
    img2 = cv2.flip(image,-1)
    img_array.append(img_to_array(img2))
     #vertical flip
    img3 = cv2.flip(image,-1)
    # horizontal flip
    img3 = cv2.flip(img3,1)
    img_array.append(img_to_array(img3))
    # rotate clockwise
    img4 = cv2.rotate(image, cv2.ROTATE_90_CLOCKWISE)
    img_array.append(img_to_array(img4))
    # flip rotated image
    img5 = cv2.flip(img4,1)
    img_array.append(img_to_array(img5))
     # rotate anti clockwise
    img6 = cv2.rotate(image, cv2.ROTATE_90_COUNTERCLOCKWISE)
    img_array.append(img_to_array(img6))
```

```python
    # flip rotated image
    img7 = cv2.flip(img6,1)
    img_array.append(img_to_array(img7))


for i in tqdm(sketch_file):
    image = cv2.imread(sketch_path + '/' + i,1)

    # as opencv load image in bgr format converting it to rgb
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # resizing images
    image = cv2.resize(image, (SIZE, SIZE))

    # normalizing image
    image = image.astype('float32') / 255.0
    # appending normal sketch image
    sketch_array.append(img_to_array(image))

    #Image Augmentation
    # horizontal flip
    img1 = cv2.flip(image,1)
    sketch_array.append(img_to_array(img1))
     #vertical flip
    img2 = cv2.flip(image,-1)
    sketch_array.append(img_to_array(img2))
     #vertical flip
    img3 = cv2.flip(image,-1)
    # horizontal flip
    img3 = cv2.flip(img3,1)
    sketch_array.append(img_to_array(img3))
    # rotate clockwise
    img4 = cv2.rotate(image, cv2.ROTATE_90_CLOCKWISE)
    sketch_array.append(img_to_array(img4))
    # flip rotated image
    img5 = cv2.flip(img4,1)
    sketch_array.append(img_to_array(img5))
     # rotate anti clockwise
    img6 = cv2.rotate(image, cv2.ROTATE_90_COUNTERCLOCKWISE)
    sketch_array.append(img_to_array(img6))
    # flip rotated image
    img7 = cv2.flip(img6,1)
    sketch_array.append(img_to_array(img7))
```

```
100%|██████████| 188/188 [00:05<00:00, 36.05it/s]
100%|██████████| 188/188 [00:02<00:00, 82.54it/s]
```

```python
print("Total number of sketch images:",len(sketch_array))
print("Total number of images:",len(img_array))
```

```
Total number of sketch images: 1504
Total number of images: 1504
```

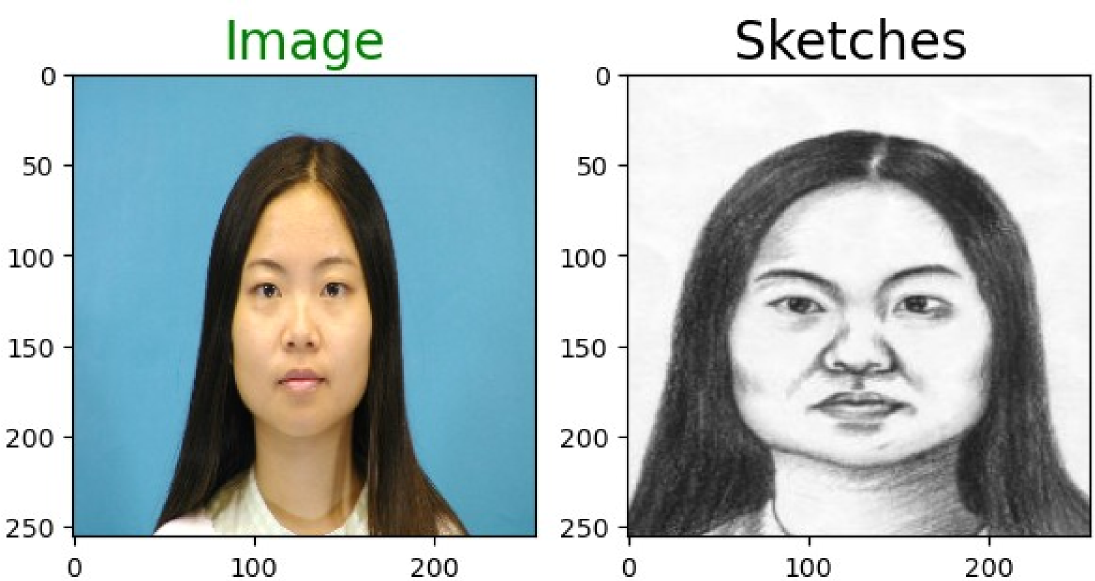
Visualizing images

Here we have plotted all augmented images and its augmented sketches

```python
# defining function to plot images pair
def plot_images(image, sketches):
    plt.figure(figsize=(7,7))
    plt.subplot(1,2,1)
    plt.title('Image', color = 'green', fontsize = 20)
    plt.imshow(image)
    plt.subplot(1,2,2)
    plt.title('Sketches ', color = 'black', fontsize = 20)
    plt.imshow(sketches)

    plt.show()

ls = [i for i in range(0,65,8)]
for i in ls:
    plot_images(img_array[i],sketch_array[i])
```



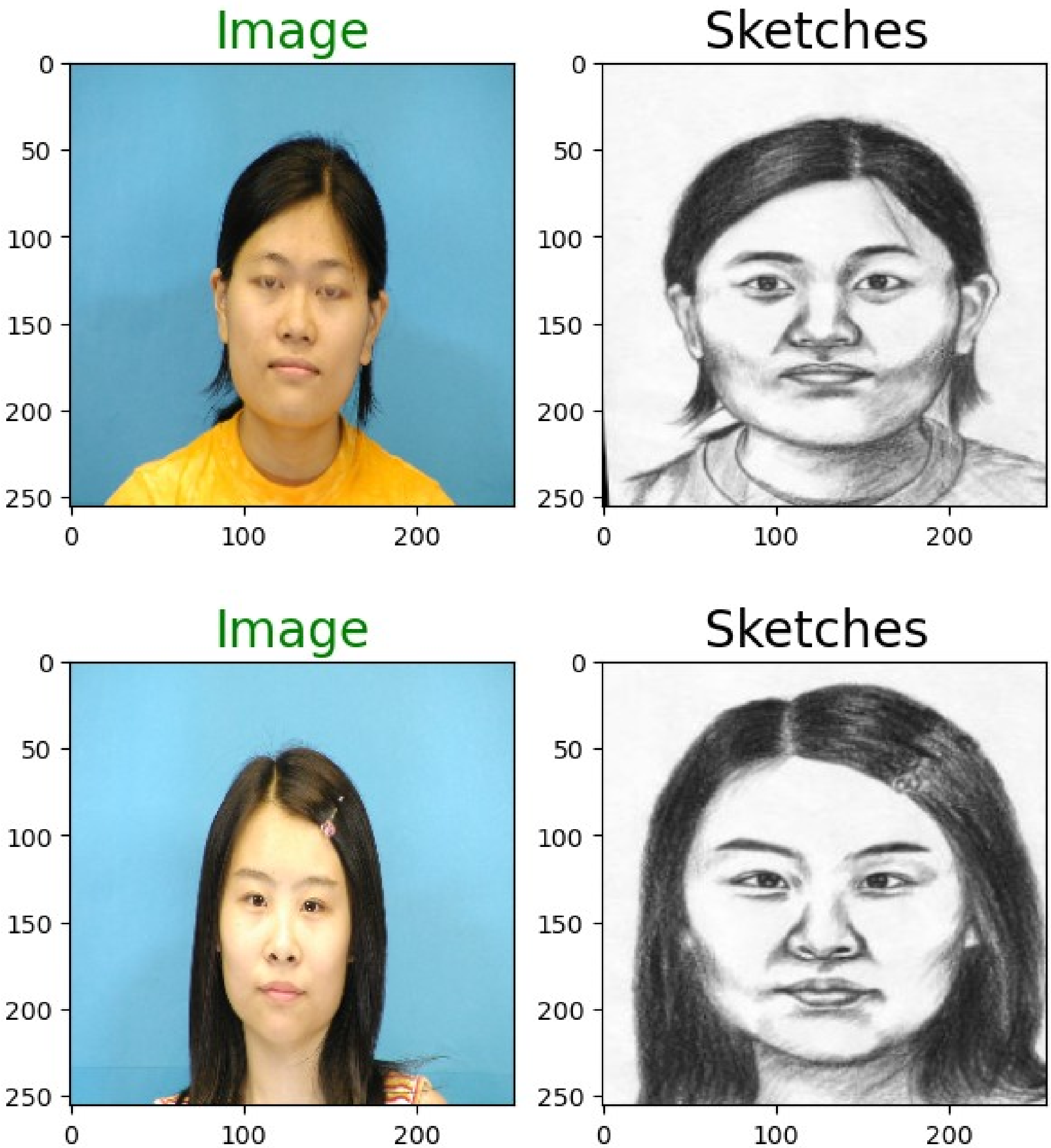


Image
Sketches
0
50
100
150
200
250
0
100
200
Image
Sketches
0
50
100
150
200
250
0
100
200

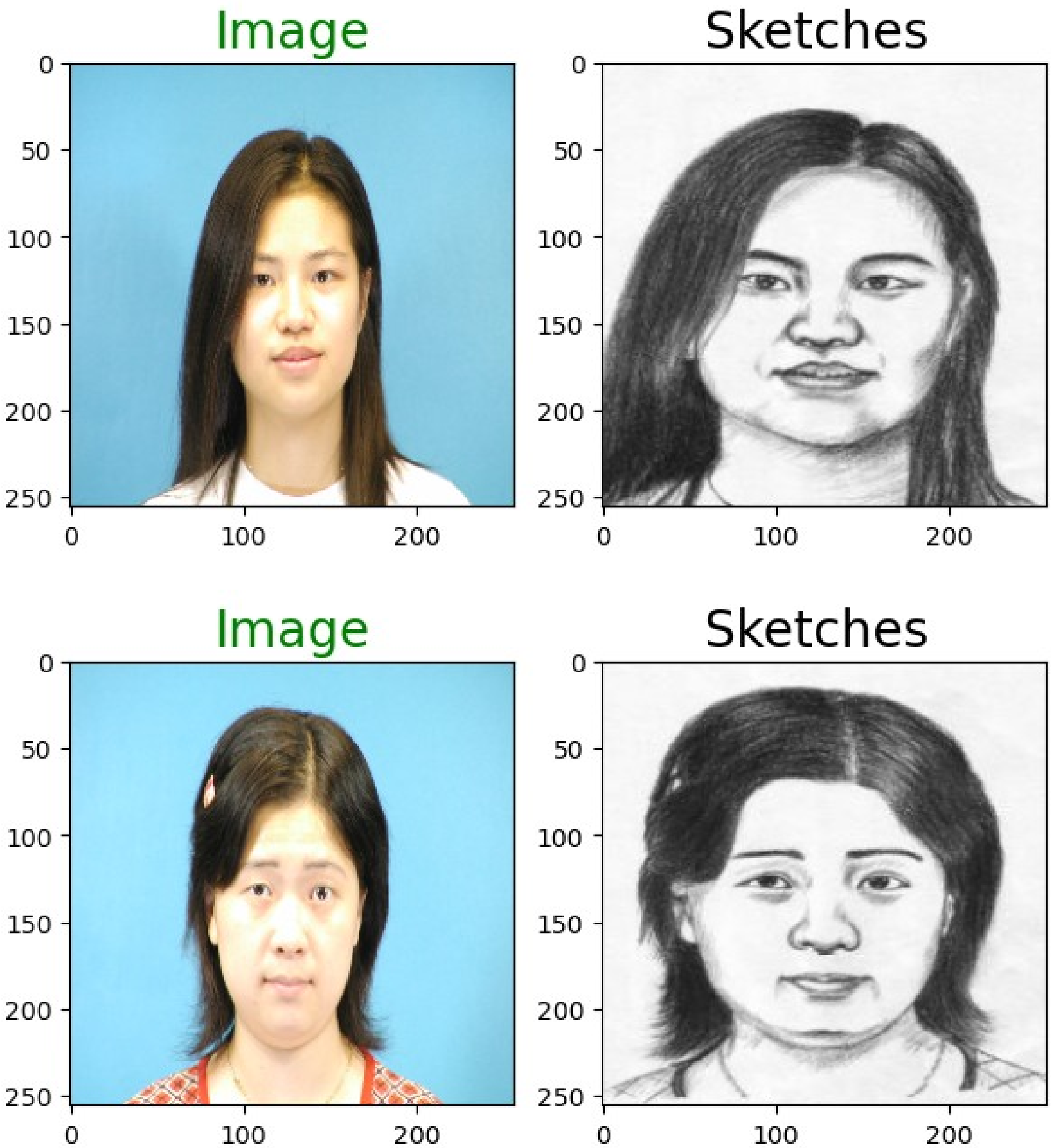


Image
Sketches
0
50
100
150
200
250
0
100
200
Image
Sketches
0
50
100
150
200
250
0
100
200

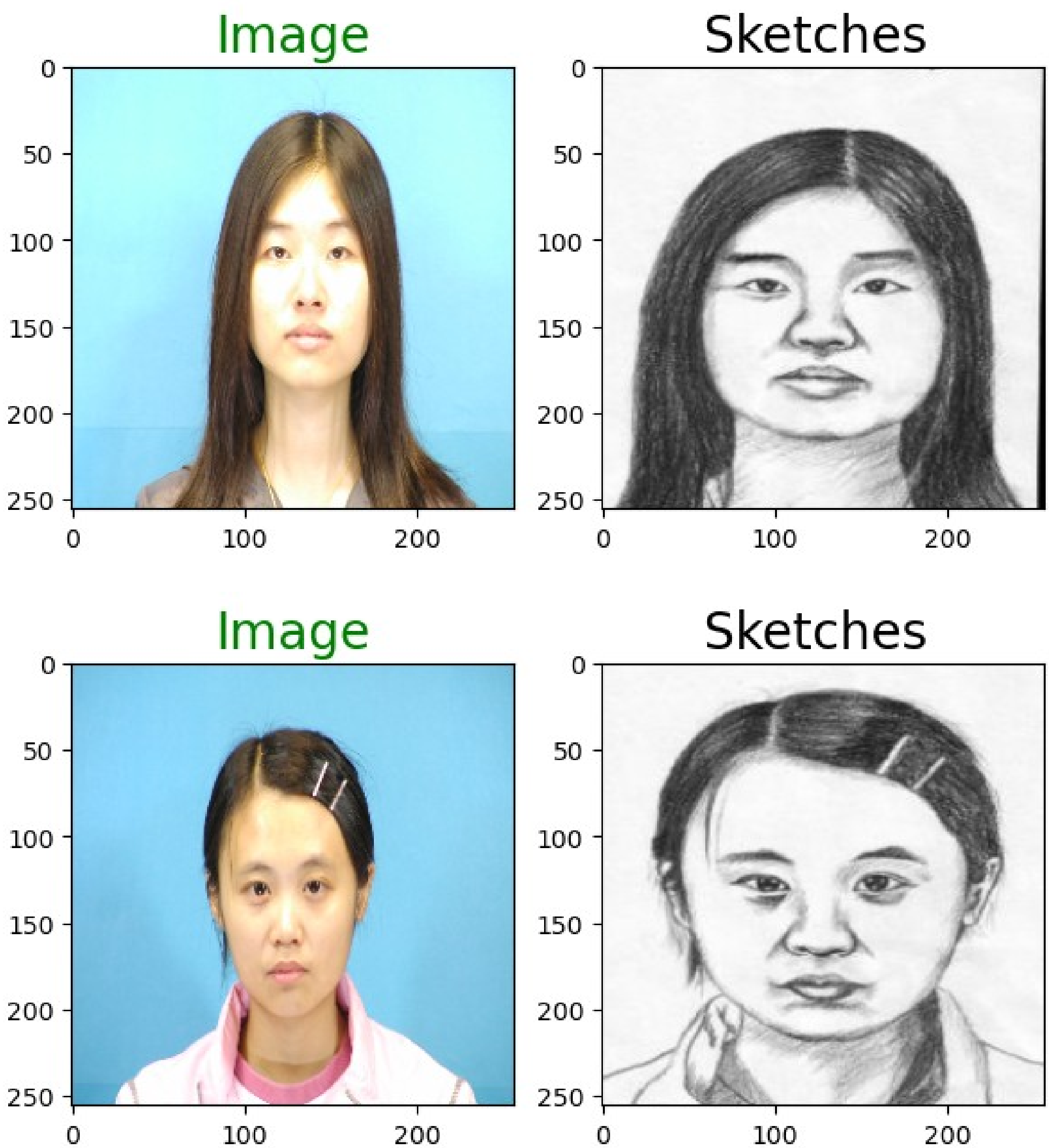
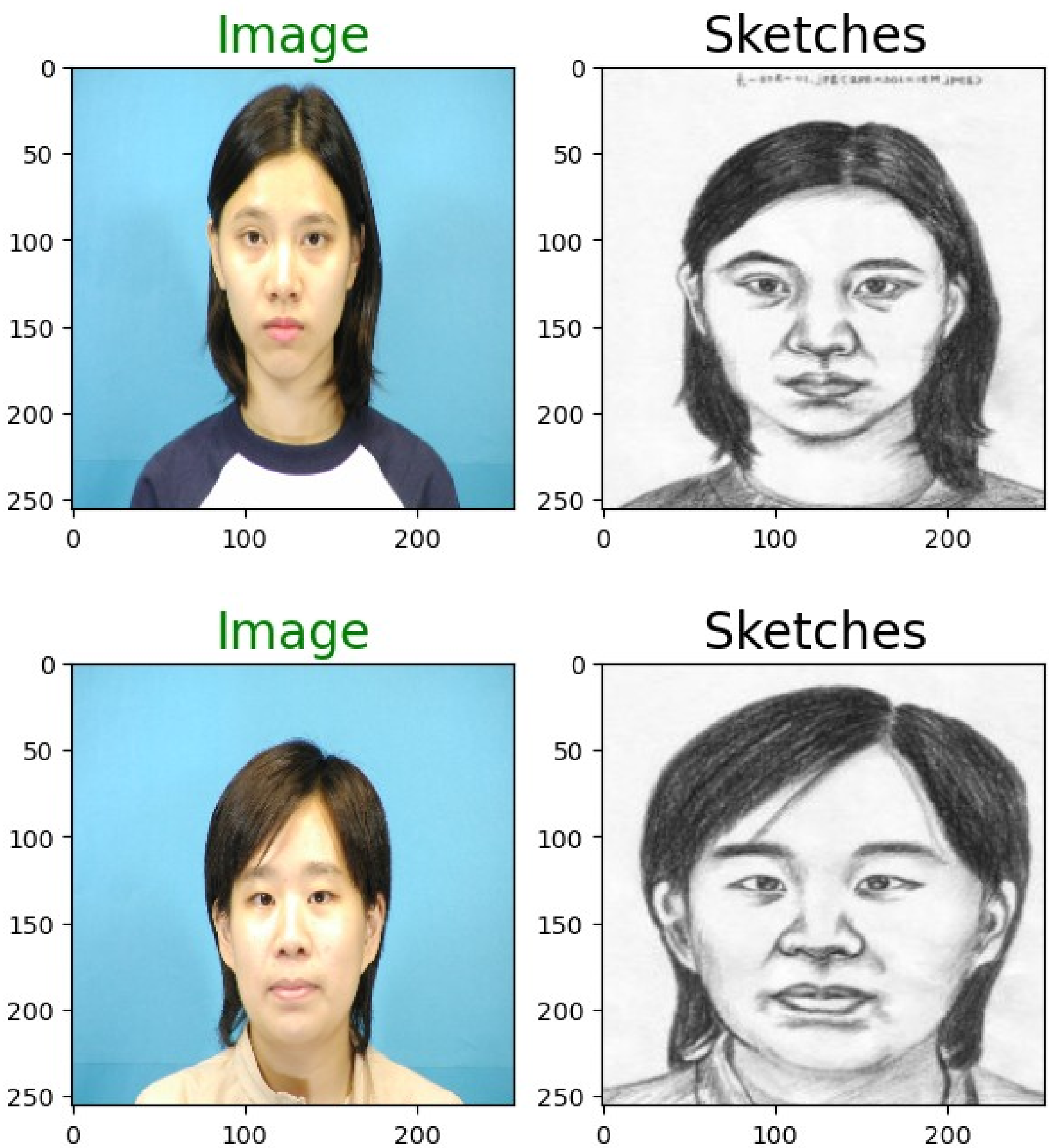


Image
Sketches
0
50
100
150
200
250
0
100
200
Image
Sketches
0
50
100
150
200
250
0
100
200



Slicing and reshaping

Out of 1504 images We have sliced them to two part. train images consist 1400 images while test images contains 104 images. After slicing image array, we reshaped them so that images can be fed directly into our encoder network

```
train_sketch_image = sketch_array[:1400]
train_image = img_array[:1400]
test_sketch_image = sketch_array[1400:]
test_image = img_array[1400:]
# reshaping
train_sketch_image = np.reshape(train_sketch_image,
(len(train_sketch_image),SIZE,SIZE,3))
```

```
train_image = np.reshape(train_image, (len(train_image),SIZE,SIZE,3))
print('Train color image shape:',train_image.shape)
test_sketch_image = np.reshape(test_sketch_image,
(len(test_sketch_image),SIZE,SIZE,3))
test_image = np.reshape(test_image, (len(test_image),SIZE,SIZE,3))
print('Test color image shape',test_image.shape)
```

```
Train color image shape: (1400, 256, 256, 3)
Test color image shape (104, 256, 256, 3)
```

Downsample layer

```
def downsample(filters, size, apply_batch_normalization = True):
    downsample = tf.keras.models.Sequential()
    downsample.add(keras.layers.Conv2D(filters = filters, kernel_size
= size, strides = 2, use_bias = False, kernel_initializer =
'he_normal'))
    if apply_batch_normalization:
        downsample.add(keras.layers.BatchNormalization())
    downsample.add(keras.layers.LeakyReLU())
    return downsample
```

Upsample Layer

```
def upsample(filters, size, apply_dropout = False):
    upsample = tf.keras.models.Sequential()
    upsample.add(keras.layers.Conv2DTranspose(filters = filters,
kernel_size = size, strides = 2, use_bias = False, kernel_initializer
= 'he_normal'))
    if apply_dropout:
        upsample.add(tf.keras.layers.Dropout(0.1))
    upsample.add(tf.keras.layers.LeakyReLU())
    return upsample
```

Model

Here we have use sequence of downsample layer for encoder and upsample layer for decoder

```
def model():
    encoder_input = keras.Input(shape = (SIZE, SIZE, 3))
    x = downsample(16, 4, False)(encoder_input)
    x = downsample(32,4)(x)
    x = downsample(64,4,False)(x)
    x = downsample(128,4)(x)
    x = downsample(256,4)(x)

    encoder_output = downsample(512,4)(x)

    decoder_input = upsample(512,4,True)(encoder_output)
    x = upsample(256,4,False)(decoder_input)
```

```
    x = upsample(128,4, True)(x)
    x = upsample(64,4)(x)
    x = upsample(32,4)(x)
    x = upsample(16,4)(x)
    x = tf.keras.layers.Conv2DTranspose(8,(2,2),strides = (1,1),
padding = 'valid')(x)
    decoder_output = tf.keras.layers.Conv2DTranspose(3,(2,2),strides =
(1,1), padding = 'valid')(x)


    return tf.keras.Model(encoder_input, decoder_output)
```

```
# to get summary of model
model = model()
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 sequential (Sequential)     (None, 127, 127, 16)      768

 sequential_1 (Sequential)   (None, 62, 62, 32)        8320

 sequential_2 (Sequential)   (None, 30, 30, 64)        32768

 sequential_3 (Sequential)   (None, 14, 14, 128)       131584

 sequential_4 (Sequential)   (None, 6, 6, 256)         525312

 sequential_5 (Sequential)   (None, 2, 2, 512)         2099200

 sequential_6 (Sequential)   (None, 6, 6, 512)         4194304

 sequential_7 (Sequential)   (None, 14, 14, 256)       2097152

 sequential_8 (Sequential)   (None, 30, 30, 128)       524288

 sequential_9 (Sequential)   (None, 62, 62, 64)        131072

 sequential_10 (Sequential)  (None, 126, 126, 32)      32768

 sequential_11 (Sequential)  (None, 254, 254, 16)      8192
```

```
 conv2d_transpose_6 (Conv2DT  (None, 255, 255, 8)      520
 ranspose)

 conv2d_transpose_7 (Conv2DT  (None, 256, 256, 3)      99
 ranspose)

=================================================================
Total params: 9,786,347
Trainable params: 9,784,491
Non-trainable params: 1,856
_________________________________________________________________
```

```
# ## Model
# Here we have defined two blocks of networks. Encoder network takes
256 by 256 image and downsample it to 16 by 16 latent vector
# by passing our image via series of Convolution and Maxpooling layer.
This downsampled 16 by 16 latent vector is upsampled by passing
# through series of Convolution and UpSampling layer. The final
decoder output is same as our encoder input. This upsamples output of
decoder
# is compared with our sketches and reconstruction loss is calculated.
This loss is minimized by updating weight and bias of network through
# backpropagation.
# encoder_input = keras.Input(shape=(SIZE,SIZE, 3), name="img")
# x = Conv2D(filters = 16, kernel_size = (3,3), activation = 'relu',
padding = 'same')(encoder_input)
# x = MaxPool2D(pool_size = (2,2))(x)

# x = Conv2D(filters = 32,kernel_size = (3,3),strides = (2,2),
activation = 'relu', padding = 'valid')(x)
# x = Conv2D(filters = 64, kernel_size = (3,3), strides = (2,2),
activation = 'relu', padding = 'same')(x)
# x = MaxPool2D(pool_size = (2,2))(x)

# x = Conv2D(filters = 128, kernel_size = (3,3), activation = 'relu',
padding = 'same')(x)
# x = Conv2D(filters = 256 , kernel_size = (3,3), activation = 'relu',
padding = 'same')(x)
# encoder_output = Conv2D(filters = 512 , kernel_size = (3,3),
activation = 'relu', padding = 'same')(x)
# encoder = tf.keras.Model(encoder_input, encoder_output)

# decoder_input = Conv2D(filters = 512 ,kernel_size = (3,3),
activation = 'relu', padding = 'same')(encoder_output)
# x = UpSampling2D(size = (2,2))(decoder_input)
# x = Conv2D(filters = 256, kernel_size = (3,3), activation = 'relu',
padding = 'same')(x)
# x = Conv2D(filters = 128, kernel_size = (3,3), activation = 'relu',
padding = 'same')(x)
# x = UpSampling2D(size = (2,2) )(x)
```

```
# x = Conv2D(filters = 64, kernel_size = (3,3), activation = 'relu',
padding = 'same')(x)
# x = UpSampling2D(size = (2,2) )(x)
# x = Conv2D(filters = 32 , kernel_size = (3,3), activation = 'relu',
padding = 'same')(x)
# x = UpSampling2D(size = (2,2) )(x)

# x = Conv2D(filters = 16  , kernel_size = (3,3), activation = 'relu',
padding = 'same')(x)
# decoder_output = Conv2D(filters = 3, kernel_size = (3,3), activation
= 'relu', padding = 'same')(x)

# # final model
# model = keras.Model(encoder_input, decoder_output)
# model.summary()
```

Compiling and Fitting our model

Here we have used Adam optimizer and mean_squared_error as loss and have trained model for 100 epochs

```
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate =
0.0001), loss = 'mean_absolute_error',
              metrics = ['acc'])

model.fit(train_image, train_sketch_image, epochs = 200, verbose = 0)

<keras.callbacks.History at 0x7fb13cb86bd0>

model.save('/kaggle/working/final_model.h5')
model.save('/kaggle/working/model_3.h5')
model.save('./model_4.h5')

import pickle
pickle.dump(model,open("model.h5","wb"))

Keras weights file (<HDF5 file "variables.h5" (mode r+)>) saving:
...layers
......conv2d_transpose
.........vars
............0
............1
......conv2d_transpose_1
.........vars
............0
............1
......input_layer
.........vars
......sequential
.........layers
```

```
............conv2d
...............vars
..................0
............leaky_re_lu
...............vars
.........vars
......sequential_1
.........layers
............batch_normalization
...............vars
..................0
..................1
..................2
..................3
............conv2d
...............vars
..................0
............leaky_re_lu
...............vars
.........vars
......sequential_10
.........layers
............conv2d_transpose
...............vars
..................0
............leaky_re_lu
...............vars
.........vars
......sequential_11
.........layers
............conv2d_transpose
...............vars
..................0
............leaky_re_lu
...............vars
.........vars
......sequential_2
.........layers
............conv2d
...............vars
..................0
............leaky_re_lu
...............vars
.........vars
......sequential_3
.........layers
............batch_normalization
...............vars
..................0
..................1
```

```
..................2
..................3
............conv2d
...............vars
..................0
............leaky_re_lu
...............vars
.........vars
......sequential_4
.........layers
............batch_normalization
...............vars
..................0
..................1
..................2
..................3
............conv2d
...............vars
..................0
............leaky_re_lu
...............vars
.........vars
......sequential_5
.........layers
............batch_normalization
...............vars
..................0
..................1
..................2
..................3
............conv2d
...............vars
..................0
............leaky_re_lu
...............vars
.........vars
......sequential_6
.........layers
............conv2d_transpose
...............vars
..................0
............dropout
...............vars
............leaky_re_lu
...............vars
.........vars
......sequential_7
.........layers
............conv2d_transpose
...............vars
```

```
..................0
............leaky_re_lu
...............vars
.........vars
......sequential_8
.........layers
............conv2d_transpose
...............vars
..................0
............dropout
...............vars
............leaky_re_lu
...............vars
.........vars
......sequential_9
.........layers
............conv2d_transpose
...............vars
..................0
............leaky_re_lu
...............vars
.........vars
...metrics
......mean
.........vars
............0
............1
......mean_metric_wrapper
.........vars
............0
............1
...optimizer
......vars
.........0
.........1
.........10
.........11
.........12
.........13
.........14
.........15
.........16
.........17
.........18
.........19
.........2
.........20
.........21
.........22
.........23
```

```
.........24
.........25
.........26
.........27
.........28
.........29
.........3
.........30
.........31
.........32
.........33
.........34
.........35
.........36
.........37
.........38
.........39
.........4
.........40
.........41
.........42
.........43
.........44
.........45
.........46
.........47
.........48
.........5
.........6
.........7
.........8
.........9
...vars
Keras model archive saving:
File Name                                             Modified
Size
config.json                                    2023-03-29 12:43:43
17886
variables.h5                                   2023-03-29 12:43:44
117557856
metadata.json                                  2023-03-29 12:43:43
64
```

Evaluating our model

```
prediction_on_test_data = model.evaluate(test_image,
test_sketch_image)
print("Loss: ", prediction_on_test_data[0])
print("Accuracy: ", np.round(prediction_on_test_data[1] * 100,1))
```

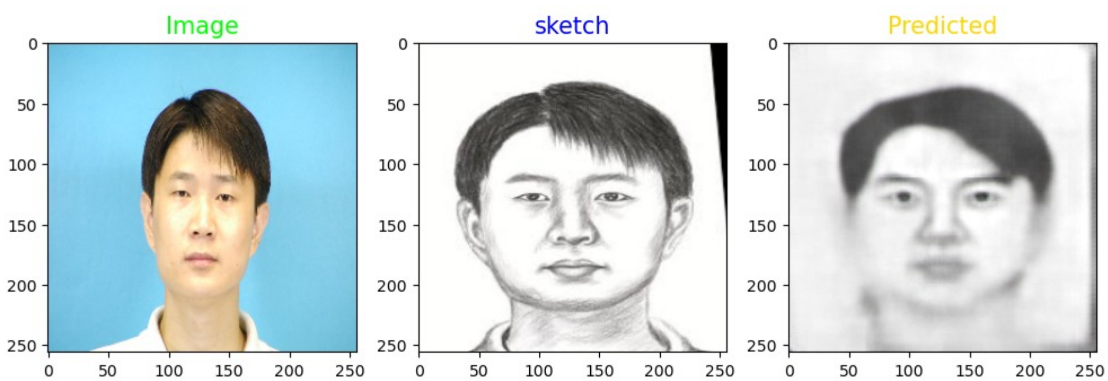
```
4/4 [==============================] - 7s 1s/step - loss: 0.1140 -
acc: 0.4082
Loss:  0.11403301358222961
Accuracy:  40.8
```

Plotting our predicted sketch along with real sketch

```
def show_images(real,sketch, predicted):
    plt.figure(figsize = (12,12))
    plt.subplot(1,3,1)
    plt.title("Image",fontsize = 15, color = 'Lime')
    plt.imshow(real)
    plt.subplot(1,3,2)
    plt.title("sketch",fontsize = 15, color = 'Blue')
    plt.imshow(sketch)
    plt.subplot(1,3,3)
    plt.title("Predicted",fontsize = 15, color = 'gold')
    plt.imshow(predicted)

ls = [i for i in range(0,95,8)]
for i in ls:
    predicted
=np.clip(model.predict(test_image[i].reshape(1,SIZE,SIZE,3)),0.0,1.0).
reshape(SIZE,SIZE,3)
    show_images(test_image[i],test_sketch_image[i],predicted)
```

```
1/1 [==============================] - 0s 336ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 85ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 102ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 117ms/step
```



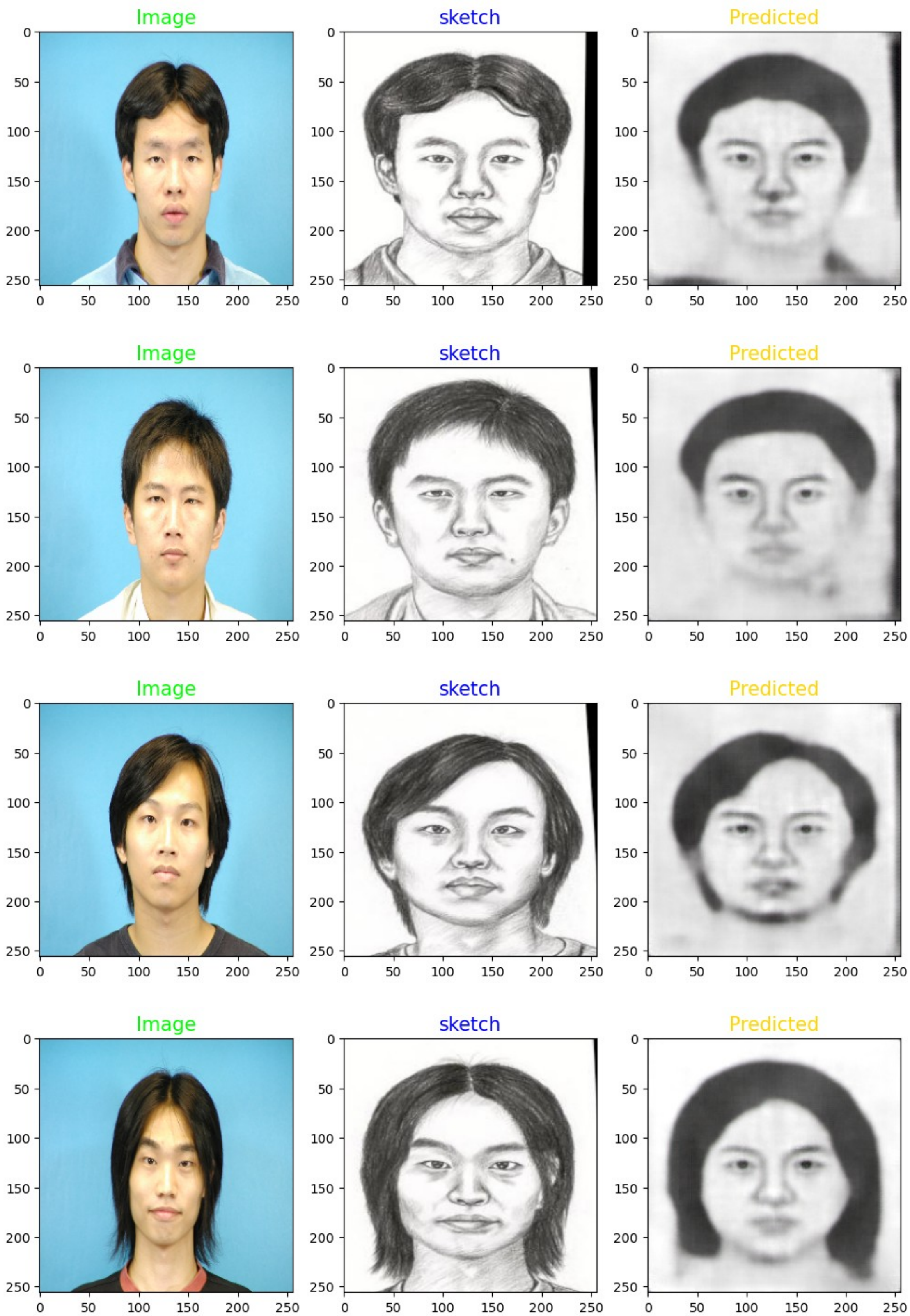


Image
sketch
Predicted
0
50
100
150
200
250
Image
sketch
Predicted
Image
sketch
Predicted
Image
sketch
Predicted

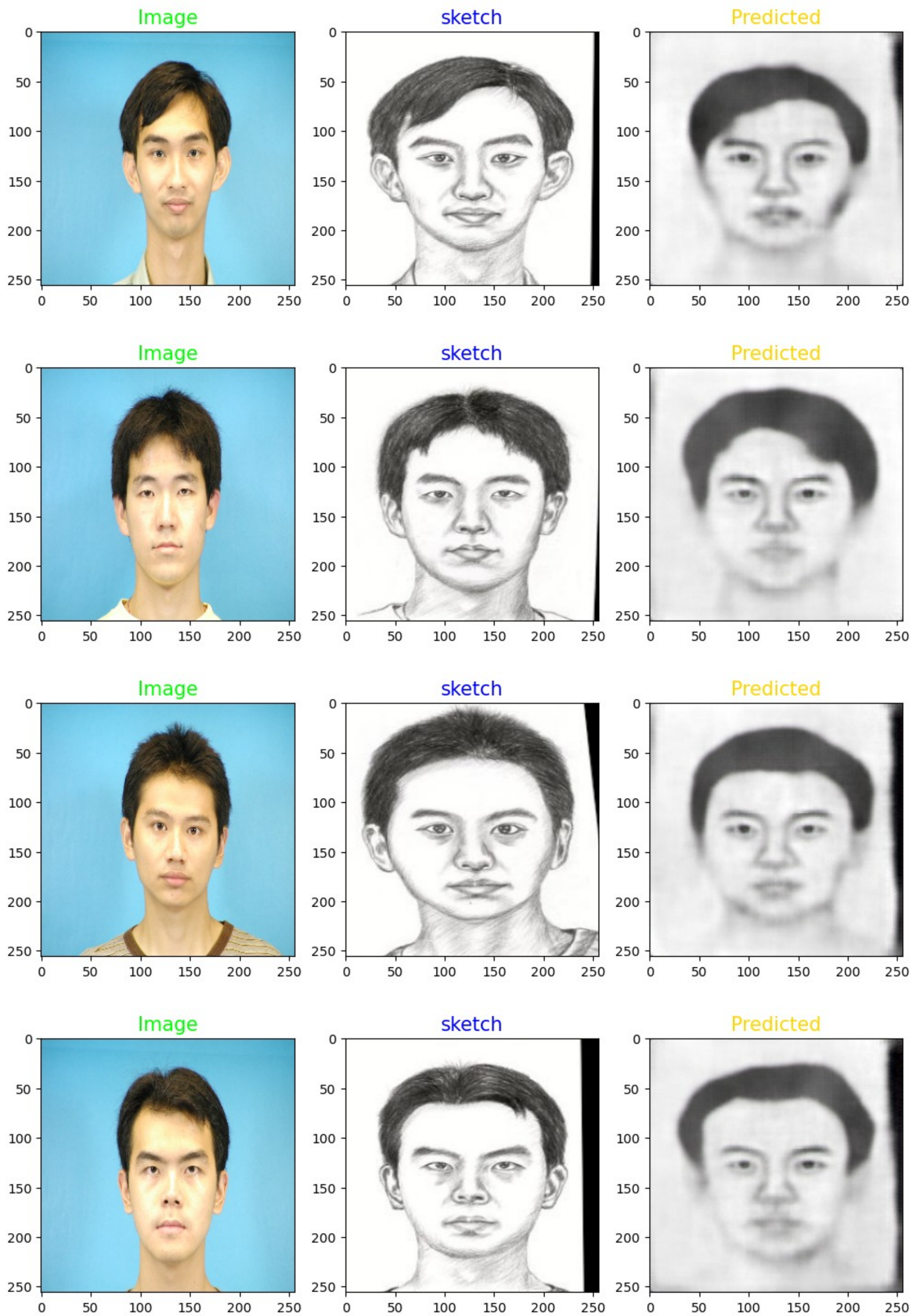


Image
sketch
Predicted
Image
sketch
Predicted
Image
sketch
Predicted
Image
sketch
Predicted
0
50
100
150
200
250

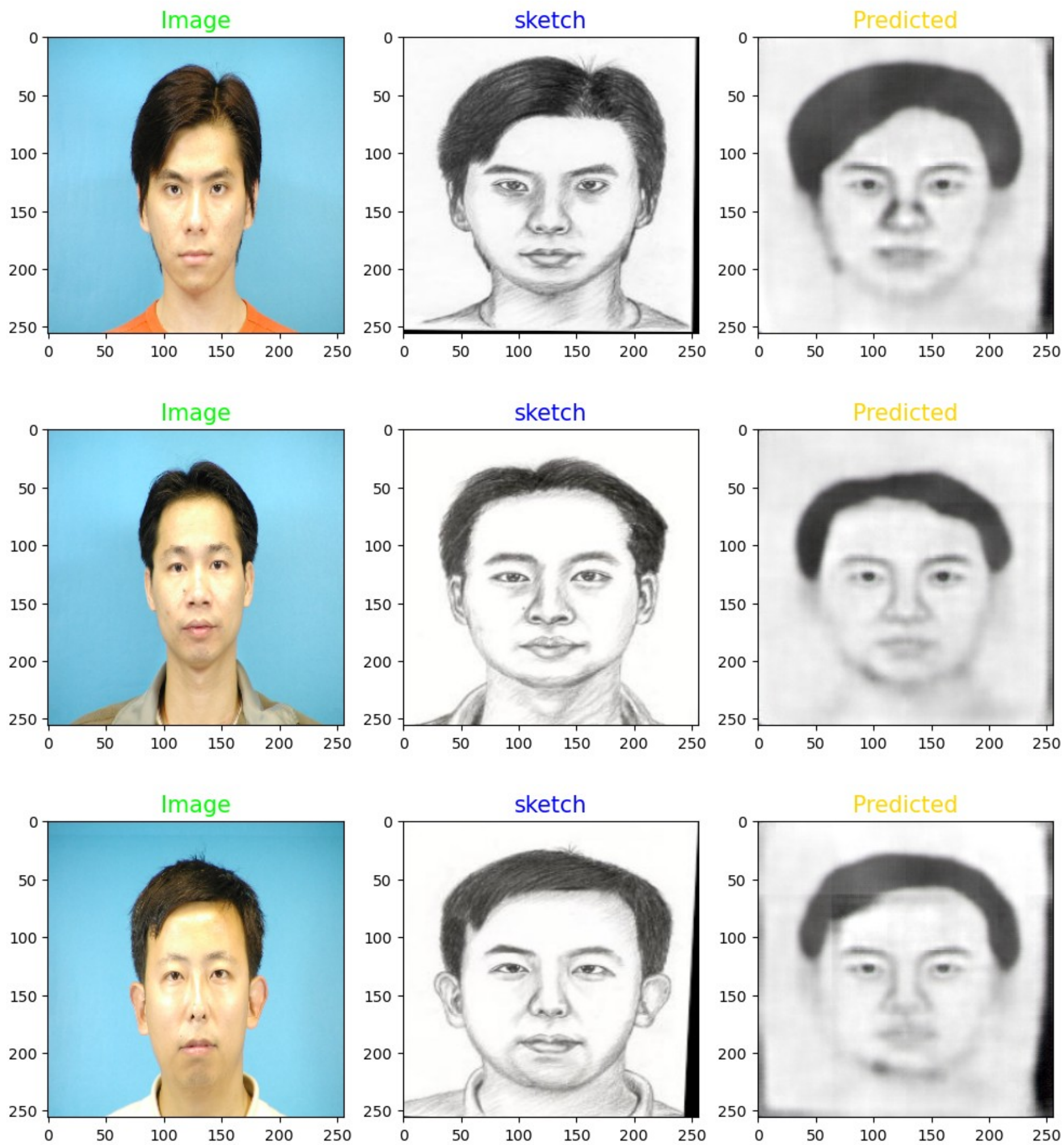


Image
sketch
Predicted
Image
sketch
Predicted
Image
sketch
Predicted
0
50
100
150
200
250