นางสาวณิชกานต์ รังษีวงศ์ 6010504741

รายงาน Class Project

1. Build a Pokemon classifier system using CNN

- ขั้นตอนแรกPreprocess data โดยทำให้รูปมีขนาด 128x128x3 และ rescaleโดยการ หาร255

```
imagedata = []
for i in range(0,len(imagelist2)):
    img = mpimg.imread(imagelist2[i]).astype('float32') / 255
    img = resize(img, (128,128,3))
    imagedata.append(img)
```

- ขั้นที่ 2 แบ่งรูปเป็น2ส่วน คือ image_train และ image_test โดยให้ image_test มี20%ของทั้งหมด โดยมีlabelว่าอยู่กลุ่มไหน

```
lb = LabelBinarizer()

imagelist1 = np.array(imagelist1)
imagelist1 = lb.fit_transform(imagelist1)
imagedata = np.array(imagedata)
print(imagedata.shape)
```

```
(6818, 128, 128, 3)
```

```
image_train, image_test, label_train, label_test = train_test_split(imagedata, imagelist1, test_size
=0.2, random_state=5)
print(image_train.shape)
print(label_train.shape)
print(image_test.shape)
print(label_test.shape)
```

```
(5454, 128, 128, 3)
(5454, 150)
(1364, 128, 128, 3)
(1364, 150)
```

- ขั้นที่ 3 สร้าง model

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(150, activation='sigmoid'),
])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 1024)              12846080
_________________________________________________________________
dense_1 (Dense)              (None, 150)               153750
=================================================================
Total params: 13,023,414
Trainable params: 13,023,414
Non-trainable params: 0
```

- ขั้นที่ 4 compile modelและ train modelโดยได้เลือกใช้ loss funtion คือ categorical_crossentropy และ optimizer คือ 'Adadelta' โดยนำเข้าmodel fit 500 epochs

```
model.compile(optimizer = 'Adadelta' , loss = tf.keras.losses.categorical_crossentropy, metrics=['accuracy'])
history = model.fit(image_train, label_train, epochs=500, validation_data=(image_test, label_test))
```

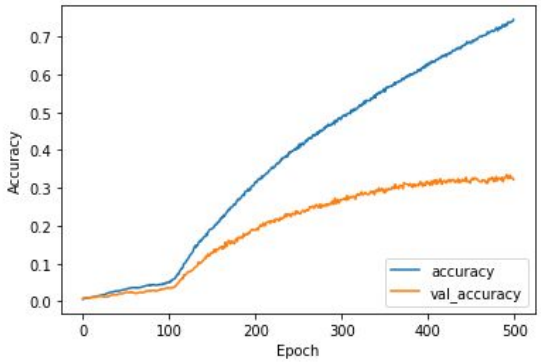
```
Train on 5454 samples, validate on 1364 samples
Epoch 1/500
5454/5454 [==============================] - 57s 11ms/sample - loss: 5.0114 - accuracy: 0.0059 - val_loss: 5.0098 - val_accuracy: 0.0073
Epoch 2/500
5454/5454 [==============================] - 56s 10ms/sample - loss: 5.0108 - accuracy: 0.0077 - val_loss: 5.0095 - val_accuracy: 0.0088
Epoch 3/500
5454/5454 [==============================] - 58s 11ms/sample - loss: 5.0102 - accuracy: 0.0090 - val_loss: 5.0092 - val_accuracy: 0.0088
Epoch 4/500
5454/5454 [==============================] - 56s 10ms/sample - loss: 5.0097 - accuracy: 0.0094 - val_loss: 5.0089 - val_accuracy: 0.0095
Epoch 5/500
```

ผลลัพธ์ คือ loss: 3.2582 - accuracy: 0.3218 มีค่า accuracyน้อยเนื่องจากtrain จำนวนครั้งน้อยเกินไปถ้าเพิ่มจำนวนครั้งในการtrainจะทำให้ค่าaccuracyเพิ่มแต่เนื่องจากอุปกรณ์ที่ใช้ไม่สามารถเพิ่มจำนวนครั้งในการtrainทำให้ผลไม่เป็นตามที่คาดหมาย

```
_loss: 3.2656 - val_accuracy: 0.3270
Epoch 500/500
5454/5454 [==============================] - 59s 11ms/sample - loss: 1.2314 - accuracy: 0.7444 - val
_loss: 3.2582 - val_accuracy: 0.3218
```

In [10]:
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(image_test,  label_test, verbose=2)
```

```
1364/1364 - 3s - loss: 3.2582 - accuracy: 0.3218
```



2. By using the same data in 1, compress Pokemon image data using Autoencoder.

- ขั้นตอนแรกPreprocess data โดยทำให้รูปมีขนาด 64x64x3 และ rescalโดยการ หาร255

```
imagedata = []
for i in range(0,len(imagelist2)):
    img = mpimg.imread(imagelist2[i]).astype('float32') / 255
    img = resize(img, (64,64,3))
    imagedata.append(img)
```

- ขั้นที่ 2 แบ่งรูปเป็น2ส่วน คือ image_train และ image_test โดยให้ image_test มี20%ของทั้งหมด

```
X_train, X_test, y_train, y_test = train_test_split(imagedata, imagelist1, test_size=0.2, random_sta
te=5)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(5468, 64, 64, 3)
(5468, 150)
(1368, 64, 64, 3)
(1368, 150)
```

- ขั้นที่ 3 สร้าง model

```
autoencoder = tf.keras.models.Sequential()

# Let's build the encoder CNN
autoencoder.add(tf.keras.layers.Conv2D(64, (3,3), strides=1, padding="same", input_shape=(64, 64,
3)))
autoencoder.add(tf.keras.layers.BatchNormalization())
autoencoder.add(tf.keras.layers.Activation('relu'))
autoencoder.add(tf.keras.layers.AveragePooling2D((2,2), padding="same"))

autoencoder.add(tf.keras.layers.Conv2D(32, (3,3), strides=1, padding="same"))
autoencoder.add(tf.keras.layers.BatchNormalization())
autoencoder.add(tf.keras.layers.Activation('relu'))
autoencoder.add(tf.keras.layers.AveragePooling2D((2,2), padding="same"))

# Let's build the decoder CNN
autoencoder.add(tf.keras.layers.BatchNormalization())
autoencoder.add(tf.keras.layers.Activation('relu'))
autoencoder.add(tf.keras.layers.UpSampling2D((2, 2)))

autoencoder.add(tf.keras.layers.Conv2D(64, (3,3), strides=1, padding="same"))
autoencoder.add(tf.keras.layers.BatchNormalization())
autoencoder.add(tf.keras.layers.Activation('relu'))
autoencoder.add(tf.keras.layers.UpSampling2D((2, 2)))


autoencoder.add(tf.keras.layers.Conv2D(3, (3,3), strides=1, activation='sigmoid', padding="same"))
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 64, 64, 64)        1792
_________________________________________________________________
batch_normalization_16 (Batc (None, 64, 64, 64)        256
_________________________________________________________________
activation_16 (Activation)   (None, 64, 64, 64)        0
_________________________________________________________________
average_pooling2d_8 (Average (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 32, 32, 32)        18464
_________________________________________________________________
batch_normalization_17 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
activation_17 (Activation)   (None, 32, 32, 32)        0
_________________________________________________________________
average_pooling2d_9 (Average (None, 16, 16, 32)        0
_________________________________________________________________
batch_normalization_18 (Batc (None, 16, 16, 32)        128
_________________________________________________________________
activation_18 (Activation)   (None, 16, 16, 32)        0
_________________________________________________________________
up_sampling2d_8 (UpSampling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 32, 32, 64)        18496
_________________________________________________________________
batch_normalization_19 (Batc (None, 32, 32, 64)        256
_________________________________________________________________
activation_19 (Activation)   (None, 32, 32, 64)        0
_________________________________________________________________
up_sampling2d_9 (UpSampling2 (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 64, 64, 3)         1731
=================================================================
Total params: 41,251
Trainable params: 40,867
Non-trainable params: 384
_________________________________________________________________
```

- ขั้นที่ 4 compile modelและ train modelโดยได้เลือกใช้ loss funtion คือ binary_crossentropyและ optimizer คือ Adam(lr=0.01) โดยนำเข้าmodel fit 500 epochs

```
autoencoder.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.01))
```

```
autoencoder.fit(X_train,
          X_train,
          epochs=100,
          batch_size=200,
          validation_data=(X_test, X_test))
```

ผลลัพธ์ คือ loss: 0.4163 และภาพที่ได้ถูก compress แล้ว

```
In [0]: predicted = autoencoder.predict(X_test[:10].reshape(-1, 64, 64, 3))

In [0]: predicted.shape

Out[0]: (10, 64, 64, 3)

In [0]: fig, axes = plt.subplots(nrows=2, ncols=10, sharex=True, sharey=True, figsize=(80,16))
        for images, row in zip([X_test[:10], predicted], axes):
            for img, ax in zip(images, row):
                ax.imshow(img.reshape((64, 64, 3)))
                ax.get_xaxis().set_visible(False)
                ax.get_yaxis().set_visible(False)
```

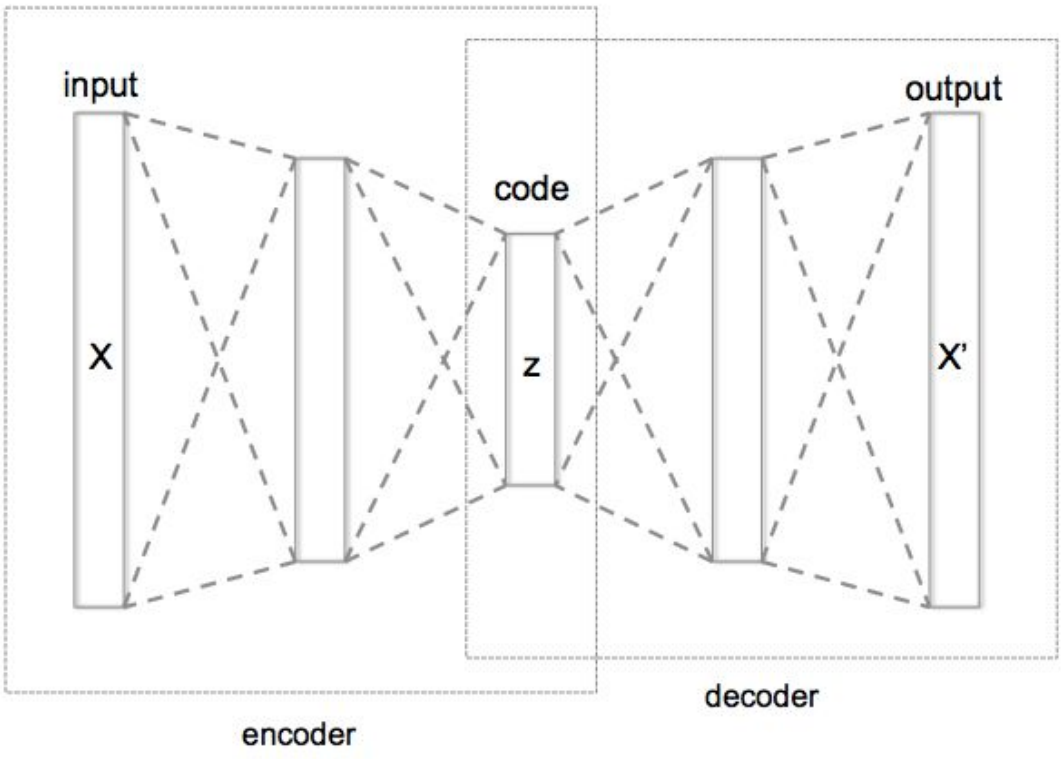
3. Anime face generation:

3.1 Use Variational Autoencoder to generate faces of anime characters.

- ขั้นที่ 1 เตรียมรูปโดยใช้ ImageDataGenerator และrescaleโดยการ หาร255

```
INPUT_DIM = (128,128,3) # Image dimension
BATCH_SIZE = 512
Z_DIM = 200 # Dimension of the latent vector (z)

data_flow = ImageDataGenerator(rescale=1./255).flow_from_directory(DATA_FOLDER,
                                                                   target_size = INPUT_DIM[:2],
                                                                   batch_size = BATCH_SIZE,
                                                                   shuffle = True,
                                                                   class_mode = 'input',
                                                                   subset = 'training'
                                                                   )
```

- ขั้นที่2 สร้างmodel โดยแบ่งเป็น 2 ส่วนคือ encoder และ decoder โดยเอาทั้งสองส่วนมาเชื่อมกัน



ส่วน encoder

```
Model: "model_1"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
encoder_input (InputLayer)   (None, 128, 128, 3)  0
_________________________________________________________________________________________________
encoder_conv_0 (Conv2D)      (None, 64, 64, 32)   896         encoder_input[0][0]
_________________________________________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 64, 64, 32)   0           encoder_conv_0[0][0]
_________________________________________________________________________________________________
encoder_conv_1 (Conv2D)      (None, 32, 32, 64)   18496       leaky_re_lu_1[0][0]
_________________________________________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 32, 32, 64)   0           encoder_conv_1[0][0]
_________________________________________________________________________________________________
encoder_conv_2 (Conv2D)      (None, 16, 16, 64)   36928       leaky_re_lu_2[0][0]
_________________________________________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 64)   0           encoder_conv_2[0][0]
_________________________________________________________________________________________________
encoder_conv_3 (Conv2D)      (None, 8, 8, 64)     36928       leaky_re_lu_3[0][0]
_________________________________________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 8, 8, 64)     0           encoder_conv_3[0][0]
_________________________________________________________________________________________________
flatten_1 (Flatten)          (None, 4096)         0           leaky_re_lu_4[0][0]
_________________________________________________________________________________________________
mu (Dense)                   (None, 200)          819400      flatten_1[0][0]
_________________________________________________________________________________________________
log_var (Dense)              (None, 200)          819400      flatten_1[0][0]
_________________________________________________________________________________________________
encoder_output (Lambda)      (None, 200)          0           mu[0][0]
                                                              log_var[0][0]
=================================================================================================
Total params: 1,732,048
Trainable params: 1,732,048
Non-trainable params: 0
```

ส่วน decoder

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
decoder_input (InputLayer)   (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              823296
_________________________________________________________________
reshape_1 (Reshape)          (None, 8, 8, 64)          0
_________________________________________________________________
decoder_conv_0 (Conv2DTransp (None, 16, 16, 64)        36928
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 16, 16, 64)        0
_________________________________________________________________
decoder_conv_1 (Conv2DTransp (None, 32, 32, 64)        36928
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 32, 32, 64)        0
_________________________________________________________________
decoder_conv_2 (Conv2DTransp (None, 64, 64, 32)        18464
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 64, 64, 32)        0
_________________________________________________________________
decoder_conv_3 (Conv2DTransp (None, 128, 128, 3)       867
_________________________________________________________________
activation_1 (Activation)    (None, 128, 128, 3)       0
=================================================================
Total params: 916,483
Trainable params: 916,483
Non-trainable params: 0
_________________________________________________________________
```

ส่วนที่เอามาเชื่อมกัน

```
vae_input = vae_encoder_input

# Output will be the output of the decoder. The term - decoder(encoder_output)
# combines the model by passing the encoder output to the input of the decoder.
vae_output = vae_decoder(vae_encoder_output)

# Input to the combined model will be the input to the encoder.
# Output of the combined model will be the output of the decoder.
vae_model = Model(vae_input, vae_output)

vae_model.summary()
```

ภาพรวมของmodel

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 128, 128, 3)	0	
encoder_conv_0 (Conv2D)	(None, 64, 64, 32)	896	encoder_input[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 64, 64, 32)	0	encoder_conv_0[0][0]
encoder_conv_1 (Conv2D)	(None, 32, 32, 64)	18496	leaky_re_lu_1[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 32, 32, 64)	0	encoder_conv_1[0][0]
encoder_conv_2 (Conv2D)	(None, 16, 16, 64)	36928	leaky_re_lu_2[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 16, 16, 64)	0	encoder_conv_2[0][0]
encoder_conv_3 (Conv2D)	(None, 8, 8, 64)	36928	leaky_re_lu_3[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 8, 8, 64)	0	encoder_conv_3[0][0]
flatten_1 (Flatten)	(None, 4096)	0	leaky_re_lu_4[0][0]
mu (Dense)	(None, 200)	819400	flatten_1[0][0]
log_var (Dense)	(None, 200)	819400	flatten_1[0][0]
encoder_output (Lambda)	(None, 200)	0	mu[0][0] log_var[0][0]
model_2 (Model)	(None, 128, 128, 3)	916483	encoder_output[0][0]

```
Total params: 2,648,531
Trainable params: 2,648,531
Non-trainable params: 0
```

- ขั้นที่ 3 compile modelและ train modelโดยได้เลือกใช้ loss funtion คือ total_loss และoptimizer คือ Adam(lr=0.0005) โดยนำเข้าmodel fit 500 epochs โดยรัน 32 epochs เพราะcomputeไม่สามารถรันได้มากกว่านี้

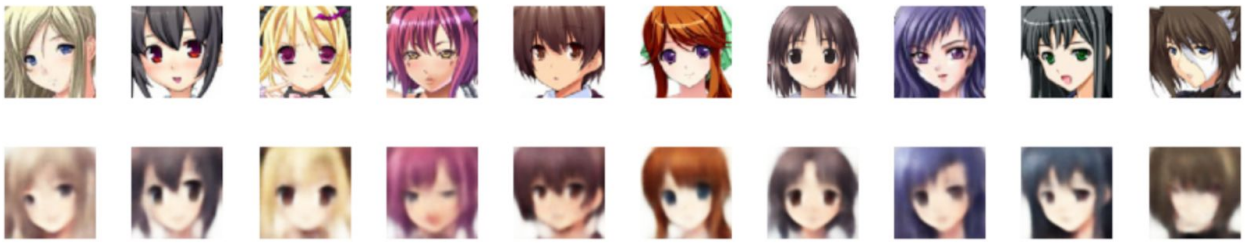
```
LEARNING_RATE = 0.0005
N_EPOCHS = 200
LOSS_FACTOR = 10000
```

```
def kl_loss(y_true, y_pred):
    kl_loss =  -0.5 * K.sum(1 + log_var - K.square(mean_mu) - K.exp(log_var), axis = 1)
    return kl_loss

def total_loss(y_true, y_pred):
    return LOSS_FACTOR*r_loss(y_true, y_pred) + kl_loss(y_true, y_pred)

def r_loss(y_true, y_pred):
    return K.mean(K.square(y_true - y_pred), axis = [1,2,3])
```

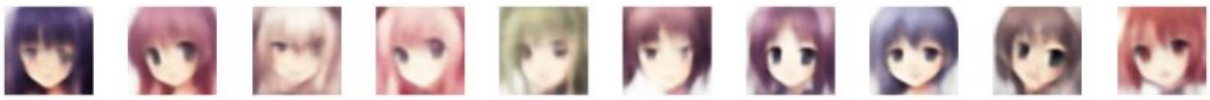
```
adam_optimizer = Adam(lr = LEARNING_RATE)

vae_model.compile(optimizer=adam_optimizer, loss = total_loss, metrics = [r_loss, kl_loss])

checkpoint_vae = ModelCheckpoint('/content/drive/My Drive/Colab Notebooks/weights.h5', save_weights_
only = True, verbose=1)
```

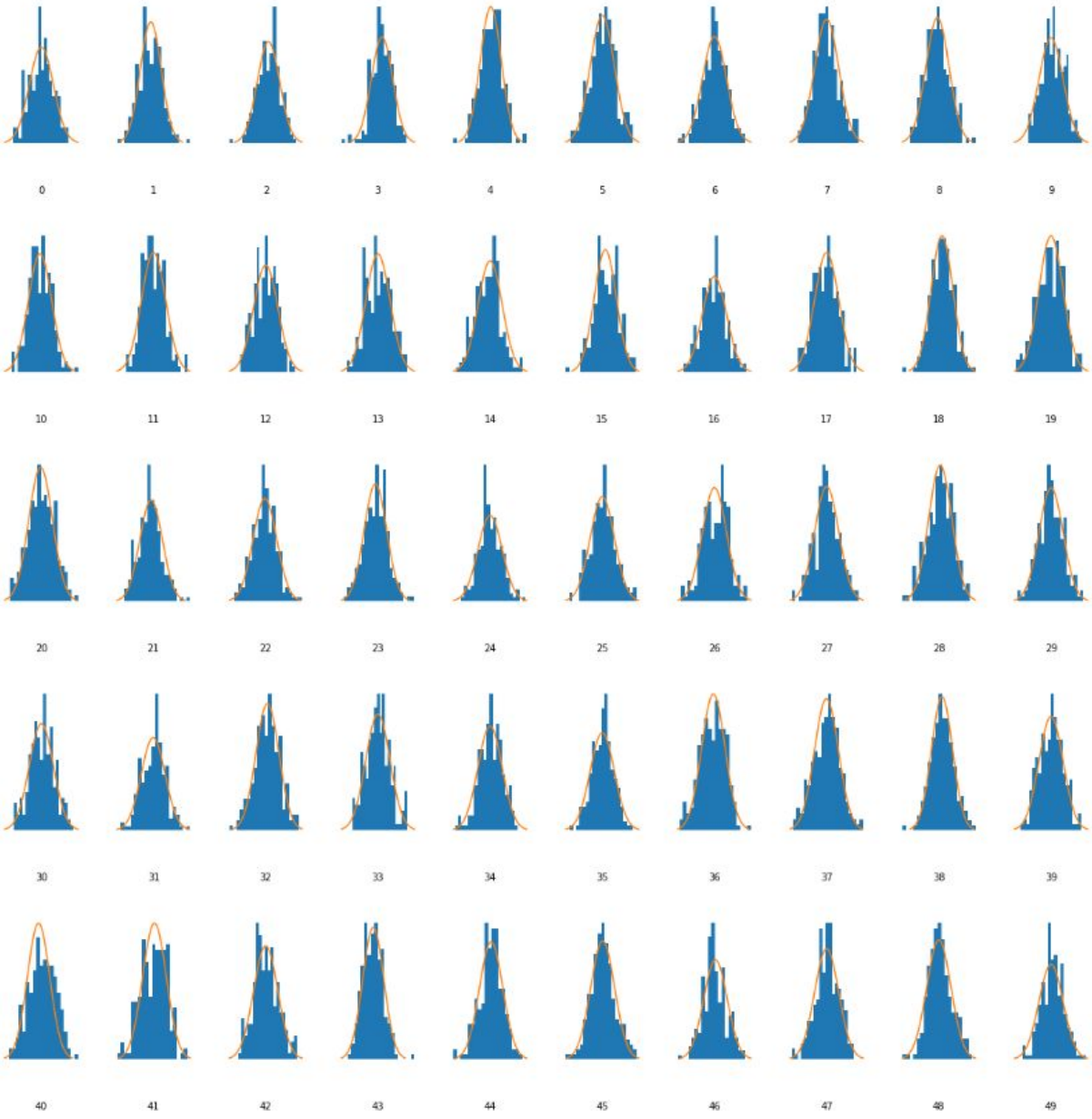
```
vae_model.fit_generator(data_flow,
                        shuffle=True,
                        epochs = 50,
                        initial_epoch = 0,
                        steps_per_epoch=NUM_IMAGES / BATCH_SIZE,
                        callbacks=[checkpoint_vae])
```

ผลลัพธ์ loss: 298.2921 - r_loss: 0.0235 - kl_loss: 62.9809 ภาพที่ได้ค่อนข้างเบลอ เนื่องจากจำนวนครั้งในการ train น้อยเกินไปแต่เห็นได้ว่าภาหน้าคนมีการเปลี่ยนแปลง



```
vae_generate_images(n_to_show=10)
```



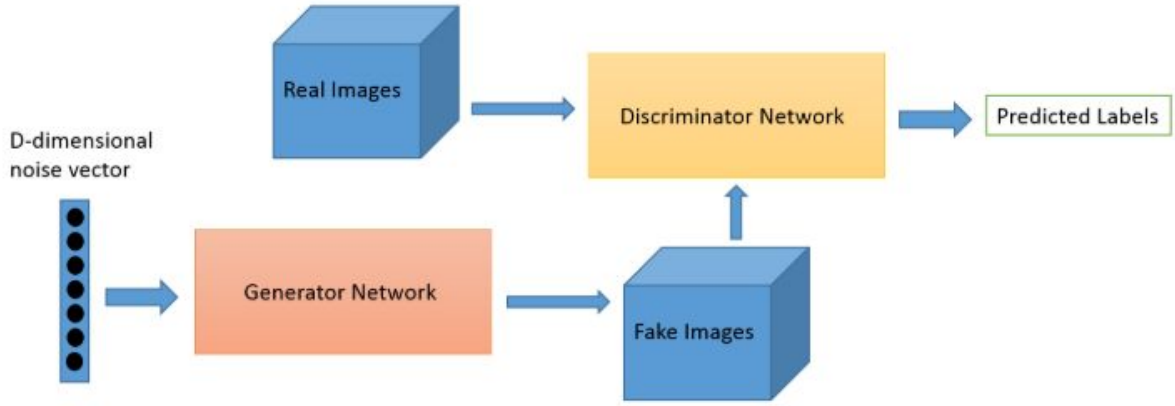


3.2 Use GAN to generate faces of anime characters.

- ขั้นที่ 1 เตรียมรูปโดยใช้ ImageDataGenerator และrescaleโดยการ หาร255

```
DATA_FOLDER = '/content/data'
INPUT_DIM = (64,64,3) # Image dimension
BATCH_SIZE = 63565
Z_DIM = 200 # Dimension of the latent vector (z)
epochs=6000
#Loading the data
batch_size=64
batch_count = 63565 / batch_size

train_data = ImageDataGenerator(rescale=1./255).flow_from_directory(DATA_FOLDER,
                                                                    target_size = INPUT_DIM[:2],
                                                                    batch_size = BATCH_SIZE,
                                                                    shuffle = True,
                                                                    class_mode = 'input',
                                                                    subset = 'training'
                                                                    )
data = next(train_data)
data = data[0]
batch_size=64
```

- ขั้นที่2 สร้างmodel โดยแบ่งเป็น 2 ส่วนคือ Generatorและ Discriminator โดยเอาทั้งสองส่วนมาเชื่อมกันเป็น gan



ส่วนGenerator ได้ผลเป็นfake image

```
def Generator():
    generator=Sequential(name='Generator')
    generator.add(Dense(units=256,input_dim=25, name='gen_input'))
    generator.add(LeakyReLU(0.2))
    generator.add(BatchNormalization(momentum=0.8))

    generator.add(Dense(units=512))
    generator.add(LeakyReLU(0.2))
    generator.add(BatchNormalization(momentum=0.8))

    generator.add(Dense(units=1024))
    generator.add(LeakyReLU(0.2))
    generator.add(BatchNormalization(momentum=0.8))

    generator.add(Dense(units=12288, activation='tanh', name='gen_output'))

    generator.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0002,beta_1=0.5))
    return generator
```

```
Model: "Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gen_input (Dense)            (None, 256)               6656
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dense_4 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dense_5 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
batch_normalization_5 (Batch (None, 1024)              4096
_________________________________________________________________
gen_output (Dense)           (None, 12288)             12595200
=================================================================
Total params: 13,265,920
Trainable params: 13,262,336
Non-trainable params: 3,584
```

ส่วน Discriminator

```
def Discriminator():
    discriminator=Sequential(name='Discriminator')
    discriminator.add(Dense(units=1024,input_dim=12288, name='disc_input'))

    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.2))

    discriminator.add(Dense(units=512))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.2))

    discriminator.add(Dense(units=256))
    discriminator.add(LeakyReLU(0.2))

    discriminator.add(Dense(units=128))
    discriminator.add(LeakyReLU(0.2))

    discriminator.add(Dense(units=1, activation='sigmoid', name='disc_output'))

    discriminator.compile(loss='binary_crossentropy',  optimizer=Adam(learning_rate=0.0002,beta_1=0.
5))
    return discriminator
```

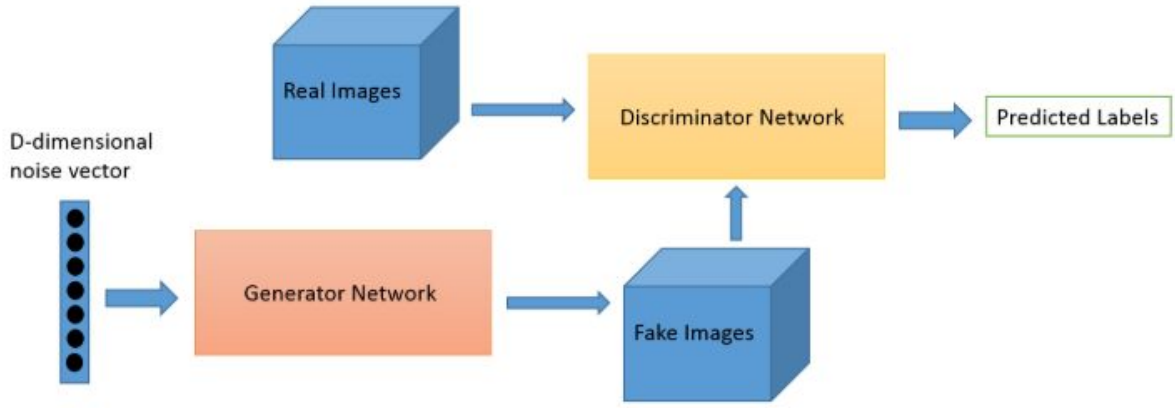
```
_________________________________________________________________
Model: "Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
disc_input (Dense)           (None, 1024)              12583936
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
disc_output (Dense)          (None, 1)                 129
=================================================================
Total params: 13,273,089
Trainable params: 13,273,089
Non-trainable params: 0
```

ส่วนที่นำ Generator และ Discriminator มาเชื่อมกัน

```
generator=Generator()
generator.summary()
discriminator =Discriminator()
discriminator.summary()
discriminator.trainable=False
gan_input = Input(shape=(25,), name='GAN_input')
gan= Model(gan_input, discriminator(generator(gan_input)) , name='GAN_model')
gan.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0002,beta_1=0.5))
gan.summary()
```

```
_________________________________________________________________
Model: "GAN_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
GAN_input (InputLayer)       [(None, 25)]              0
_________________________________________________________________
Generator (Sequential)       (None, 12288)             13265920
_________________________________________________________________
Discriminator (Sequential)   (None, 1)                 13273089
=================================================================
Total params: 26,539,009
Trainable params: 13,262,336
Non-trainable params: 13,276,673
_________________________________________________________________
```

- ขั้นที่ 3 compile modelและ train modelโดยได้เลือกใช้ loss funtion คือ binary_crossentropy และ optimizer คือ Adam(lr=0.0002) โดยนำไปcomplieโดยตอนแรกสร้างnoise vector แล้วนำเข้าGenerator Network แล้วจะได้ Fake images จากนั้นนำ Real Image และ Fake Images รวมกัน จากนั้นนำเข้า Discriminator Network เพื่อ train Discriminator Network จากนั้นสร้าง noise vector และy_ganเพื่อนำเข้า GAN เพื่อ train GAN

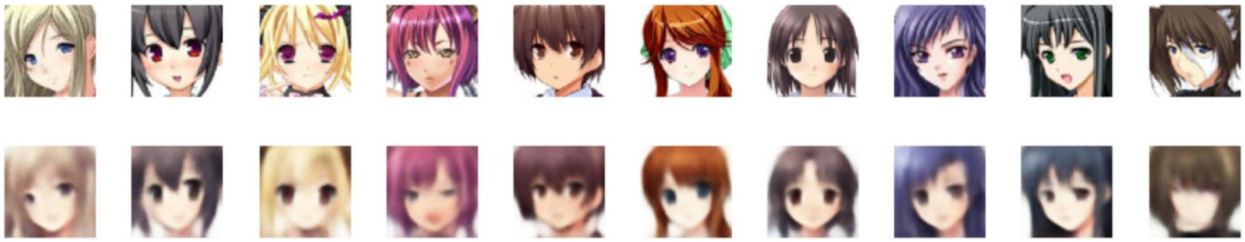


```
examples=25
dim=(5,5)
figsize=(30,30)
for e in range(1,6000+1 ):
    print("Epoch %d" %e)
    for _ in tqdm(range(batch_size)):
        noise= np.random.normal(0,1, [batch_size, 25])

        generated_images = generator.predict(noise)
        image_batch = []
        for c in range(64):
            image_batch.append(data[np.random.randint(low=0,high=63565)])
        image_batch = np.array(image_batch)
        image_batch = image_batch.reshape(64,12288)
        X= np.concatenate((image_batch, generated_images))

        y_dis=np.zeros(2*batch_size)
        y_dis[:batch_size]=0.9

        discriminator.trainable=True
        discriminator.train_on_batch(X, y_dis)

        noise= np.random.normal(0,1, [batch_size, 25])
        y_gen = np.ones(batch_size)

        discriminator.trainable=False

        gan.train_on_batch(noise, y_gen)

    if e == 1 or e % 100 == 0:
        noise= np.random.normal(loc=0, scale=1, size=[examples, 25])
        generated_images = generator.predict(noise).reshape(25,64,64,3)
        plt.figure(figsize=figsize)
        for i in range(generated_images.shape[0]):
            plt.subplot(dim[0], dim[1], i+1)
            plt.imshow(generated_images[i], interpolation='nearest')
            plt.axis('off')
        plt.tight_layout()
        plt.savefig('/content/drive/My Drive/Colab Notebooks/gan_gen_image/gan_generated_image%d.pn
g' %e)
```

ผลลัพธ์ใช้ GAN ในการ generate faces of anime characters. ภาพที่ได้จะเป็นจุดๆ เนื่องจากจำนวนครั้งในการ train น้อยเกินไป แต่หน้าของanime charactersก็มีลักษณะเปลี่ยนไปเช่นตาซ้ายกับตาขวามีลักษณะแตกต่างกัน

Compare the performance of these two models.

- ใช้ Variational Autoencoder ในการ generate faces of anime characters. ภาพที่ได้ จะเป็นภาพที่เบลอ ใช้จำนวนครั้งในการtrainน้อยกว่า เวลาที่ใช้ต่อ 1 epoch มากกว่า



- ใช้ GAN ในการ generate faces of anime characters. ภาพที่ได้จะเป็นจุดๆ ใช้จำนวนครั้งในการtrainมากกว่า เวลาที่ใช้ต่อ 1 epoch น้อยกว่า