Denoising Autoencoder using Cifar100 Dataset

Name: Fady Essam Fathy **ID**: 20190370

Name: Belal Ashraf **ID**: 20190137

First, we load the cifar100 dataset and reshaping the train and test data as the following:

Import Dataset

In [1]:
```
from keras.datasets import cifar100
import numpy
import numpy as np
import keras
import matplotlib.pyplot as plt
from keras import layers
```

In [2]:
```
(x_train, _), (x_test, _) = cifar100.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 32, 32, 3))
x_test = np.reshape(x_test, (len(x_test), 32, 32, 3))

x_train= x_train[:5000]
x_test= x_test[6000:7000]
```

In [3]:
```
print(x_train.shape)
print(x_test.shape)
```

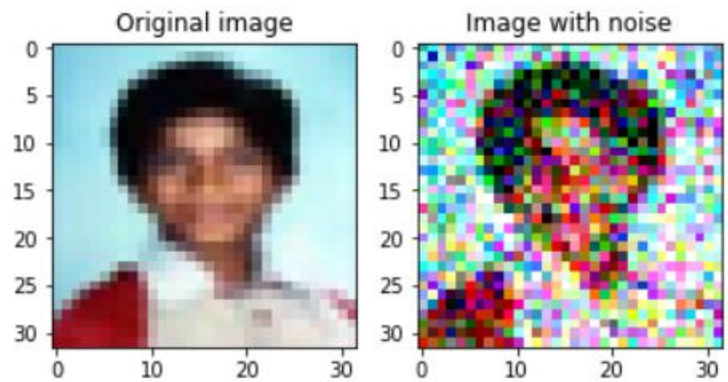
```
(5000, 32, 32, 3)
(1000, 32, 32, 3)
```

And we generate random noise with 0.3 noise factor to all images for train and test using gaussian (normal) distribution

In [4]:
```
# Generate Random Noise (normal (Gaussian) distribution)
noise_factor = 0.3
x_train_noisy = x_train + noise_factor * numpy.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * numpy.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
x_train_noisy = numpy.clip(x_train_noisy, 0., 1.)
x_test_noisy = numpy.clip(x_test_noisy, 0., 1.)
```

In [5]:
```
idx = 3
plt.subplot(1,2,1)
plt.imshow(x_train[idx].reshape(32,32,3))
plt.title('Original image')
plt.subplot(1,2,2)
plt.imshow(x_train_noisy[idx].reshape(32,32,3))
plt.title('Image with noise')
plt.show()
```

Noisy image example:



Case 1:

We add the noisy images as the input for the model training

Case 1

```
In [8]: # Add noisy Images as the Inout
```

```
In [9]: autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
        autoencoder.fit(x_train_noisy, x_train, epochs=30, batch_size=128, shuffle=True, validation_data=(x_test_noisy, x_test))
```

```
Epoch 1/30
40/40 [==============================] - 12s 253ms/step - loss: 0.6481 - val_loss: 0.6037
Epoch 2/30
40/40 [==============================] - 10s 241ms/step - loss: 0.5834 - val_loss: 0.5702
Epoch 3/30
40/40 [==============================] - 10s 256ms/step - loss: 0.5665 - val_loss: 0.5639
Epoch 4/30
40/40 [==============================] - 10s 254ms/step - loss: 0.5617 - val_loss: 0.5607
Epoch 5/30
40/40 [==============================] - 10s 252ms/step - loss: 0.5602 - val_loss: 0.5587
Epoch 6/30
40/40 [==============================] - 10s 254ms/step - loss: 0.5577 - val_loss: 0.5578
Epoch 7/30
40/40 [==============================] - 10s 248ms/step - loss: 0.5578 - val_loss: 0.5668
Epoch 8/30
40/40 [==============================] - 10s 254ms/step - loss: 0.5572 - val_loss: 0.5578
Epoch 9/30
40/40 [==============================] - 11s 279ms/step - loss: 0.5550 - val_loss: 0.5576
Epoch 10/30
40/40 [==============================] - 11s 275ms/step - loss: 0.5555 - val_loss: 0.5545
Epoch 11/30
```

And separate the encoder vector to be able to display z values

```
In [11]: encoder = keras.Model(input_img, encoded)
         encoded_imgs = encoder.predict(x_test_noisy)
         print(encoded_imgs.shape)
```

And then assign the weights for the decoder from the original autoencoder model

```
In [13]: k = 0
         for i in range (10,16):
             decoder.weights[k].assign(autoencoder.weights[i])
             k = k+1
         decoded_imgs = decoder.predict(encoded_imgs)
         print(decoded_imgs.shape)
```

<u>And the Decoded Images for Case 1:</u>

Case 2:

We add the original images as the input for the training model

```
In [16]: autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
         autoencoder.fit(x_train, x_train, epochs=30, batch_size=128, shuffle=True, validation_data=(x_test, x_test))
```

```
Epoch 1/30
40/40 [==============================] - 13s 278ms/step - loss: 0.6160 - val_loss: 0.5614
Epoch 2/30
40/40 [==============================] - 10s 262ms/step - loss: 0.5537 - val_loss: 0.5494
Epoch 3/30
40/40 [==============================] - 11s 287ms/step - loss: 0.5473 - val_loss: 0.5469
Epoch 4/30
40/40 [==============================] - 12s 303ms/step - loss: 0.5457 - val_loss: 0.5457
Epoch 5/30
40/40 [==============================] - 12s 290ms/step - loss: 0.5442 - val_loss: 0.5448
Epoch 6/30
40/40 [==============================] - 11s 278ms/step - loss: 0.5434 - val_loss: 0.5439
Epoch 7/30
40/40 [==============================] - 13s 336ms/step - loss: 0.5427 - val_loss: 0.5424
Epoch 8/30
```

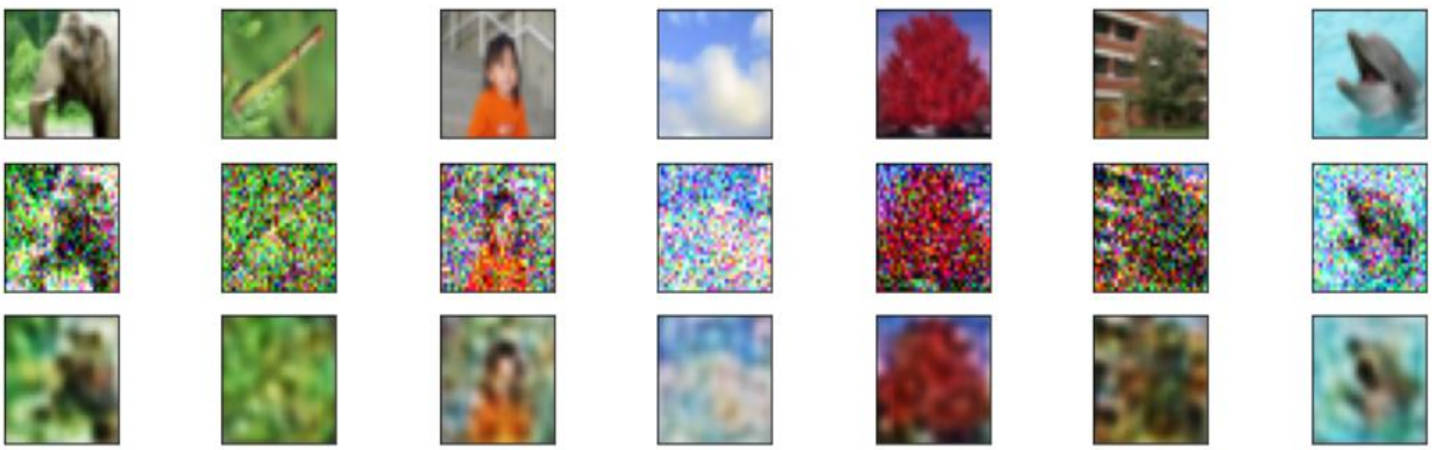
But we add the noisy images to the encoded vector as the following:

```
In [17]: encoder = keras.Model(input_img, encoded)
         encoded_imgs = encoder.predict(x_test_noisy)
         print(encoded_imgs.shape)

         32/32 [==============================] - 0s 10ms/step
         (1000, 8, 8, 32)
```

And then assign the weights for the decoder from the original autoencoder model (as case 1)

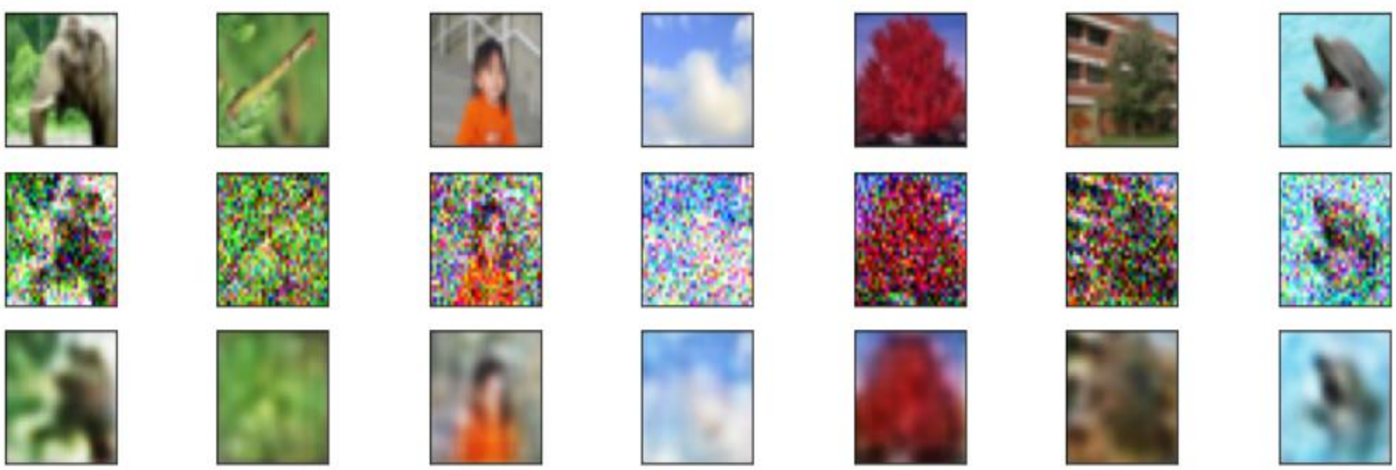
And the Decoded Images for Case 2:

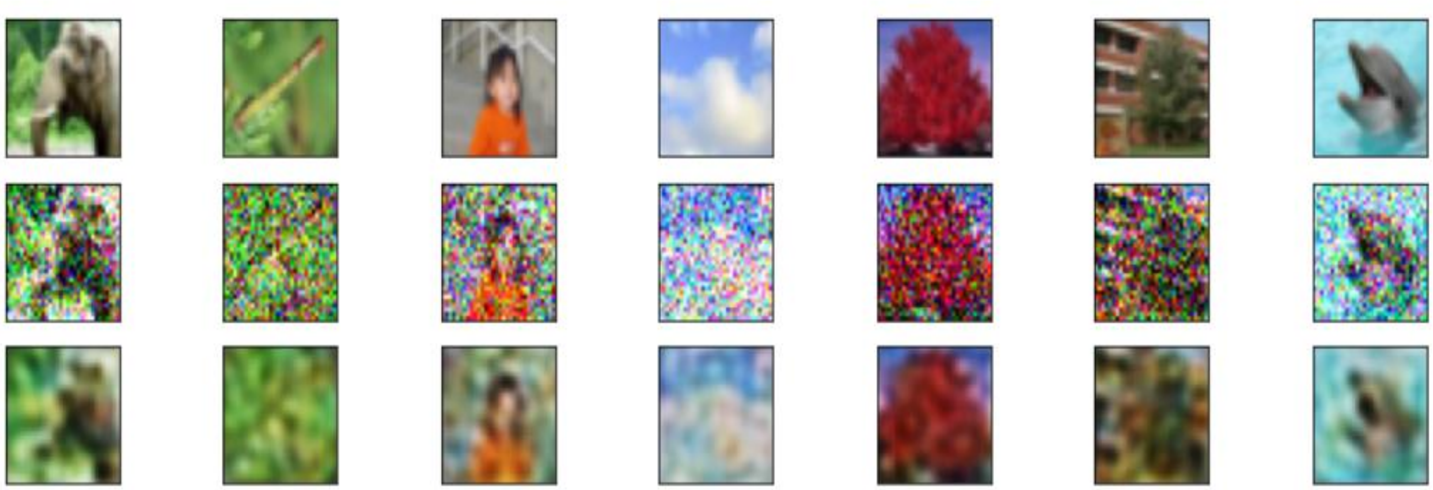


Conclusion:

As we see, when we add noisy images for fitting the model in case 1, the model could remove more noises than case 2 and the images is quite as the same (not 100%)
When case 2 made the decoded images has some noises and couldn’t remove it.

And this is the output for 2 cases together:





When applying PCA to the same noisy images:

with 200 components

with 2000 components:

with 100 components:

As we see, best case is with 100 components only

But it didn't remove almost of noises from the images so autoencoder is significantly better.