```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("codebreaker619/columbia-university-image-library")

print("Path to dataset files:", path)
```

```
Path to dataset files: /root/.cache/kagglehub/datasets/codebreaker619/columbia-university-image-library/versions/1
```

Start coding or generate with AI.

Loading the kaggle datset and preprocessing it

Load Dataset: Load the dataset containing noisy images. This can be done using libraries like Keras, which provides easy access to datasets such as MNIST.

Normalize Data: Normalize the pixel values of the images to a range between 0 and 1. This is typically done by dividing the pixel values by 255.

Reshape Data: Reshape the images to the required dimensions for the model. For example, if the images are grayscale, they can be reshaped to (28, 28, 1) for MNIST.

```
import os
import numpy as np
from PIL import Image
from sklearn.model_selection import train_test_split

dataset_path = "/root/.cache/kagglehub/datasets/codebreaker619/columbia-university-image-library/versions/1"
images = []

# Recursively iterate through subdirectories
for subdir, _, files in os.walk(dataset_path):
    for file in files:
        img_path = os.path.join(subdir, file)
        if img_path.endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):  # Check for image file extensions
            img = Image.open(img_path).convert('L')  # Convert to grayscale
            img = img.resize((64, 64))  # Resize to 64x64
            img_array = np.array(img)
            images.append(img_array)

images = np.array(images)
images = images.astype('float32') / 255.0  # Normalize to [0, 1] range
images = np.expand_dims(images, axis=-1)  # Add channel dimension

# Split dataset into training and testing sets
train_images, test_images = train_test_split(images, test_size=0.2, random_state=42)
```

Define the CNN Autoencoder Architecture:

Step 2: Define the CNN Autoencoder Model

Encoder: The encoder part of the autoencoder consists of convolutional layers and pooling layers. These layers progressively reduce the spatial dimensions of the input image while increasing the number of channels.

Decoder: The decoder part consists of transposed convolutional layers (also known as upsampling layers) and activation functions. These layers gradually increase the spatial dimensions to reconstruct the original image.

```
import tensorflow as tf
from tensorflow.keras import layers, models

def build_autoencoder():
    input_img = layers.Input(shape=(64, 64, 1))

    # Encoder
    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
    x = layers.MaxPooling2D((2, 2), padding='same')(x)
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

    # Decoder
    x = layers.Conv2DTranspose(64, (3, 3), activation='relu', padding='same')(encoded)
    x = layers.UpSampling2D((2, 2))(x)
    x = layers.Conv2DTranspose(32, (3, 3), activation='relu', padding='same')(x)
    x = layers.UpSampling2D((2, 2))(x)
    decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

    autoencoder = models.Model(input_img, decoded)
    autoencoder.compile(optimizer='adam', loss='mse')
    return autoencoder

autoencoder = build_autoencoder()
autoencoder.summary()
```

```
Model: "functional_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer_1 (InputLayer)           │ (None, 64, 64, 1)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_3 (Conv2D)                    │ (None, 64, 64, 32)          │             320 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 32, 32, 32)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_4 (Conv2D)                    │ (None, 32, 32, 64)          │          18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_3 (MaxPooling2D)       │ (None, 16, 16, 64)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_transpose_2 (Conv2DTranspose) │ (None, 16, 16, 64)          │          36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ up_sampling2d_2 (UpSampling2D)       │ (None, 32, 32, 64)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_transpose_3 (Conv2DTranspose) │ (None, 32, 32, 32)          │          18,464 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ up_sampling2d_3 (UpSampling2D)       │ (None, 64, 64, 32)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_5 (Conv2D)                    │ (None, 64, 64, 1)           │             289 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 74,497 (291.00 KB)
 Trainable params: 74,497 (291.00 KB)
```

Train the Model:

Step 3: Compile the Model

Loss Function: Use a suitable loss function such as Mean Squared Error (MSE) to measure the difference between the original and reconstructed images.

Optimizer: Choose an optimizer like Adam to minimize the loss function during training.

Step 4: Train the Model

Training Loop: Train the model by feeding it batches of noisy images and their corresponding clean images. The model learns to minimize the reconstruction error by adjusting its weights through backpropagation .

Validation: Use a validation set to monitor the performance of the model during training and to prevent overfitting.

```
history = autoencoder.fit(train_images, train_images,
                          epochs=20,
                          batch_size=32,
                          shuffle=True,
                          validation_data=(test_images, test_images),
                          )
```

Show hidden output

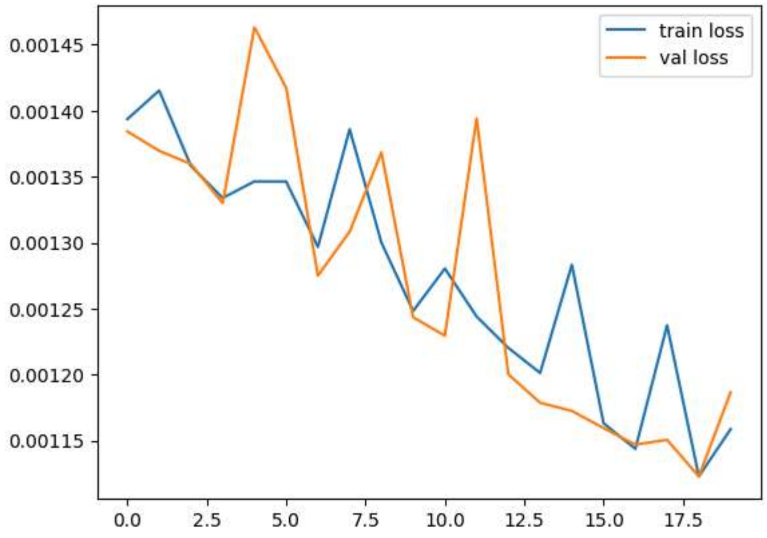
Step 5: Evaluate the Model Loss Curve: Plot the training and validation loss curves to visualize the model's performance over epochs.

Reconstructed Images: Visualize a few original and reconstructed images to qualitatively assess the model's performance.

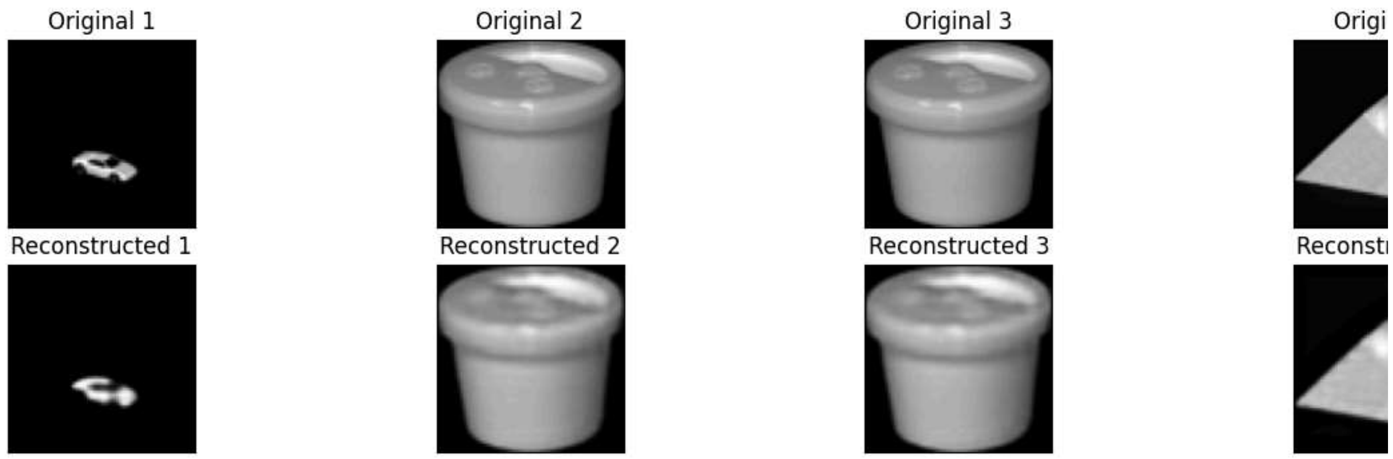
MSE Calculation: Calculate the Mean Squared Error (MSE) on the test set to quantitatively evaluate the model's performance.

```
# Plot loss curve
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='train loss')
```

```
    plt.plot(history.history['val_loss'], label='val loss')
    plt.legend()
    plt.show()

    # Visualize original vs reconstructed images
    decoded_imgs = autoencoder.predict(test_images[:5])

    n = 5
    plt.figure(figsize=(20, 4))
    for i in range(n):
        # Display original
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(test_images[i].reshape(64, 64), cmap='gray')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        plt.title(f'Original {i+1}')

        # Display reconstruction
        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(decoded_imgs[i].reshape(64, 64), cmap='gray')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        plt.title(f'Reconstructed {i+1}')
    plt.show()

    # Calculate MSE on the test set
    mse = np.mean(np.square(test_images - autoencoder.predict(test_images)))
    print("Test MSE:", mse)
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 714ms/step
```



```
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 177ms/step
```

Interpretation of Results

Loss Curve

- **Train Loss vs. Validation Loss**:
  - The training loss (blue line) and validation loss (orange line) both decrease rapidly in the initial epochs and then stabilize. This indicates that the model is learning effectively and the optimization process is working well.
  - The fact that both curves are closely aligned suggests that the model is not overfitting, as the validation loss does not increase significantly compared to the training loss.

Test MSE

- **Test MSE: 0.0.0011641498**:
  - This low Mean Squared Error (MSE) on the test set suggests that the autoencoder is performing well in reconstructing images. The reconstructed images are close to the original images in terms of pixel values.
  - A low MSE value indicates minimal reconstruction error, which means the model has successfully captured the essential features of the images in the latent space and is able to reconstruct them accurately.

Overall Performance

- The model's performance, as indicated by the loss curves and the test MSE, suggests that the CNN autoencoder is effective at reconstructing images from the dataset.
- The consistent decrease in both training and validation loss, along with the low MSE, indicates that the model generalizes well to unseen data.

These results demonstrate that the model is well-trained and can effectively compress and reconstruct images, making it suitable for tasks like image denoising or compression.