```
import numpy as np
import matplotlib.pyplot as plt
import os
from PIL import Image

from keras import layers
from keras.models import Model


def load_images_from_folder(folder):
    """Loads images from a specified folder and returns them as a numpy array."""
    images = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = Image.open(img_path).convert("RGB")  # Ensure images are RGB
        img = img.resize((32, 32))  # Resize images if needed
        images.append(np.array(img))
    return np.array(images)


def preprocess(array):
    """Normalizes the supplied array."""
    array = array.astype("float32") / 255.0
    return array


def noise(array):
    """Adds random noise to each image in the supplied array."""
    noise_factor = 0.4
    noisy_array = array + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=array.shape)
    return np.clip(noisy_array, 0.0, 1.0)


def display(array1, array2):
    """Displays ten random images from each array."""
    n = 10
    indices = np.random.randint(len(array1), size=n)
    images1 = array1[indices, :]
    images2 = array2[indices, :]

    plt.figure(figsize=(20, 4))
    for i, (image1, image2) in enumerate(zip(images1, images2)):
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(image1)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(image2)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

    plt.show()


# Prepare the data
train_folder_path = '/content/drive/MyDrive/India2_modified/train/images'
test_folder_path = '/content/drive/MyDrive/India2_modified/test/images'


# Load train and test images
train_data = load_images_from_folder(train_folder_path)
test_data = load_images_from_folder(test_folder_path)


# Normalize the data
train_data = preprocess(train_data)
test_data = preprocess(test_data)


# Create a copy of the data with added noise
noisy_train_data = noise(train_data)
noisy_test_data = noise(test_data)


# Display the original train images and the noisy versions
print("Displaying training images:")
display(train_data, noisy_train_data)
```

Displaying training images:

```
# Display the original test images and the noisy versions
print("Displaying test images:")
display(test_data, noisy_test_data)
```

Displaying test images:

```
'''Building the autoencoder
We are going to use the Functional API to build our convolutional autoencoder.'''

# Define the input shape based on the resized images
input_shape = (32, 32, 3)  # Change to (height, width, channels) for RGB images
input = layers.Input(shape=input_shape)

# Encoder
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(input)
x = layers.MaxPooling2D((2, 2), padding="same")(x)
x = layers.Conv2D(64, (3, 3), activation="relu", padding="same")(x)
x = layers.MaxPooling2D((2, 2), padding="same")(x)

# Decoder
x = layers.Conv2DTranspose(64, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2D(3, (3, 3), activation="sigmoid", padding="same")(x)  # Output for RGB images

# Autoencoder
autoencoder = Model(input, x)
autoencoder.compile(optimizer="adam", loss="binary_crossentropy")
autoencoder.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
conv2d (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_transpose (Conv2DTranspose)	(None, 16, 16, 64)	36,928
conv2d_transpose_1 (Conv2DTranspose)	(None, 32, 32, 32)	18,464
conv2d_2 (Conv2D)	(None, 32, 32, 3)	867

Total params: 75,651 (295.51 KB)

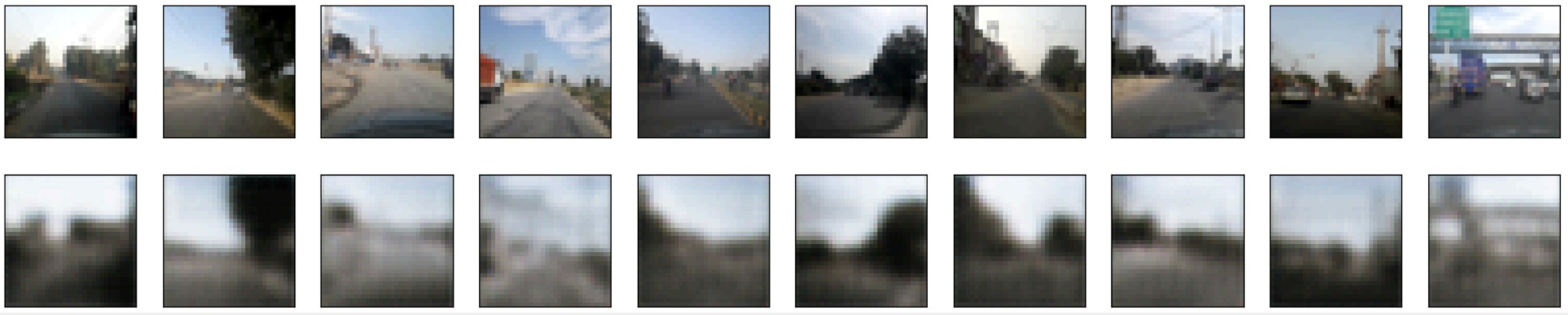
```
# Train the autoencoder with original training data
autoencoder.fit(
    x=train_data,
    y=train_data,
    epochs=20,
    batch_size=128,
    shuffle=True,
    validation_data=(test_data, test_data),
)
```

```
Epoch 1/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 6s 713ms/step - loss: 0.5570 - val_loss: 0.5460
Epoch 2/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 5s 507ms/step - loss: 0.5527 - val_loss: 0.5429
Epoch 3/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 5s 479ms/step - loss: 0.5502 - val_loss: 0.5401
Epoch 4/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 773ms/step - loss: 0.5497 - val_loss: 0.5385
Epoch 5/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 8s 474ms/step - loss: 0.5453 - val_loss: 0.5374
Epoch 6/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 704ms/step - loss: 0.5451 - val_loss: 0.5363
Epoch 7/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 9s 500ms/step - loss: 0.5448 - val_loss: 0.5351
Epoch 8/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 6s 644ms/step - loss: 0.5459 - val_loss: 0.5344
Epoch 9/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 9s 472ms/step - loss: 0.5437 - val_loss: 0.5341
Epoch 10/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 6s 698ms/step - loss: 0.5426 - val_loss: 0.5334
Epoch 11/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 9s 488ms/step - loss: 0.5397 - val_loss: 0.5325
Epoch 12/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 683ms/step - loss: 0.5436 - val_loss: 0.5319
Epoch 13/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 9s 489ms/step - loss: 0.5411 - val_loss: 0.5315
Epoch 14/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 6s 695ms/step - loss: 0.5398 - val_loss: 0.5311
Epoch 15/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 8s 477ms/step - loss: 0.5408 - val_loss: 0.5306
Epoch 16/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 724ms/step - loss: 0.5403 - val_loss: 0.5304
Epoch 17/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 5s 485ms/step - loss: 0.5427 - val_loss: 0.5298
Epoch 18/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 5s 470ms/step - loss: 0.5414 - val_loss: 0.5295
Epoch 19/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 8s 789ms/step - loss: 0.5388 - val_loss: 0.5299
Epoch 20/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 8s 490ms/step - loss: 0.5425 - val_loss: 0.5300
<keras.src.callbacks.history.History at 0x78e388efe0e0>
```

```
# Make predictions on the test data
predictions = autoencoder.predict(test_data)

# Display the original test images and the reconstructed images
display(test_data, predictions)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
```



```
# Train the autoencoder with noisy training data
autoencoder.fit(
    x=noisy_train_data,
    y=train_data,  # Use original train data as target
    epochs=20,
    batch_size=128,
    shuffle=True,
    validation_data=(noisy_test_data, test_data),  # Use noisy test data for validation
)
```

```
Epoch 1/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 5s 568ms/step - loss: 0.6344 - val_loss: 0.5907
Epoch 2/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 6s 628ms/step - loss: 0.5838 - val_loss: 0.5516
Epoch 3/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 9s 520ms/step - loss: 0.5606 - val_loss: 0.5433
Epoch 4/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 696ms/step - loss: 0.5548 - val_loss: 0.5429
Epoch 5/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 8s 493ms/step - loss: 0.5516 - val_loss: 0.5409
Epoch 6/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 697ms/step - loss: 0.5509 - val_loss: 0.5398
Epoch 7/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 8s 475ms/step - loss: 0.5457 - val_loss: 0.5391
Epoch 8/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 669ms/step - loss: 0.5481 - val_loss: 0.5387
Epoch 9/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 8s 478ms/step - loss: 0.5470 - val_loss: 0.5384
Epoch 10/20
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 674ms/step - loss: 0.5486 - val_loss: 0.5381
Epoch 11/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 4s 465ms/step - loss: 0.5483 - val_loss: 0.5380
Epoch 12/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 6s 570ms/step - loss: 0.5477 - val_loss: 0.5378
Epoch 13/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 10s 491ms/step - loss: 0.5475 - val_loss: 0.5376
Epoch 14/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 710ms/step - loss: 0.5457 - val_loss: 0.5375
Epoch 15/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 8s 456ms/step - loss: 0.5479 - val_loss: 0.5373
Epoch 16/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 6s 645ms/step - loss: 0.5470 - val_loss: 0.5372
Epoch 17/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 5s 569ms/step - loss: 0.5456 - val_loss: 0.5371
Epoch 18/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 4s 486ms/step - loss: 0.5494 - val_loss: 0.5370
Epoch 19/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 7s 685ms/step - loss: 0.5465 - val_loss: 0.5369
Epoch 20/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 9s 492ms/step - loss: 0.5447 - val_loss: 0.5369
<keras.src.callbacks.history.History at 0x78e383c7f8b0>
```

```
# Make predictions on the noisy test data
predictions = autoencoder.predict(noisy_test_data)

# Display the noisy test images and the reconstructed images
display(noisy_test_data, predictions)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```